UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Construcción de un Data Warehouse para una empresa de Retail

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Armando Ávila Ventura

ASESOR DE INFORME

M.I. Aurelio Adolfo Millán Nájera

Ciudad Universitaria, Cd. Mx., 2019

Agradecimientos

A Dios.

Por darme sabiduría, fuerza, salud, motivación y fe para culminar esta etapa de mi vida. Gracias por todas las cosas en mi vida.

A mis padres.

Por su apoyo, amor, paciencia, por sus regaños, pero sobre todo por ser el gran motor en todos mis años de estudio y hacer de mí una persona de bien. Este logro es para ustedes. Nunca terminaré de agradecerles, los amo profundamente.

A mi hermano.

Por compartir esta gran etapa de mi vida, por todas tus palabras. Gracias por haber llegado a mi vida y seguir aquí.

A mis tías.

Por su ayuda, cariño, cuidados, por todas las experiencias que viví con ustedes. Gracias por todo lo que me han dado.

A mi asesor de este informe.

Por su compromiso, entrega y total guía para obtener este logro, siempre le estaré agradecido por todo el tiempo invertido. Gracias por sus atenciones y amistad.

A mis amigos y compañeros.

Por su amistad, por haber estado conmigo en todo este tiempo.

A mis profesores y a la Facultad de Ingeniería.

Por todas las enseñanzas y experiencias aprendidas.

Construcción de un Data Warehouse para una empresa de Retail

Índice

Introducción y objetivo

El presente reporte tiene el objetivo de ver cómo una empresa de Retail[1] debido a su naturaleza, necesita tener la información disponible para poder realizar una buena toma de decisiones. Por lo que el tener un Data Warehouse (DWH) actualizado y bien diseñado es parte fundamental de su operación diaria. Esto se hace mediante la Tecnología de Información (TI) que se encarga del almacenamiento, recuperación, transmisión y manipulación de datos frecuentemente utilizado en los negocios o empresas.

Una de las ramas de TI es la Inteligencia de Negocios o Business Intelligence (BI) la cual analiza la información de una empresa a fin de generar escenarios, tendencias, pronósticos y reportes que son usados por distintas áreas de una empresa hasta los directivos para poder llevar a cabo nuevas proyecciones. Sin embargo, no siempre es fácil el llevar a cabo esto, ya que las grandes empresas tienen demasiada información en varios sistemas, por lo que es necesario el poder concentrar dicha información en un repositorio, de ahí la importancia de tener un DWH.

[1] Empresa dedicada a la venta minorista. Ver anexo

Es por esto, que la empresa de Retail contrató los servicios de una empresa de consultoría para construir un DWH en Teradata, ya que el anterior sistema estaba en Oracle y se tenían varios problemas de rendimiento. En ese momento el equipo para realizar el DWH se constituía como se muestra en la figura 1.

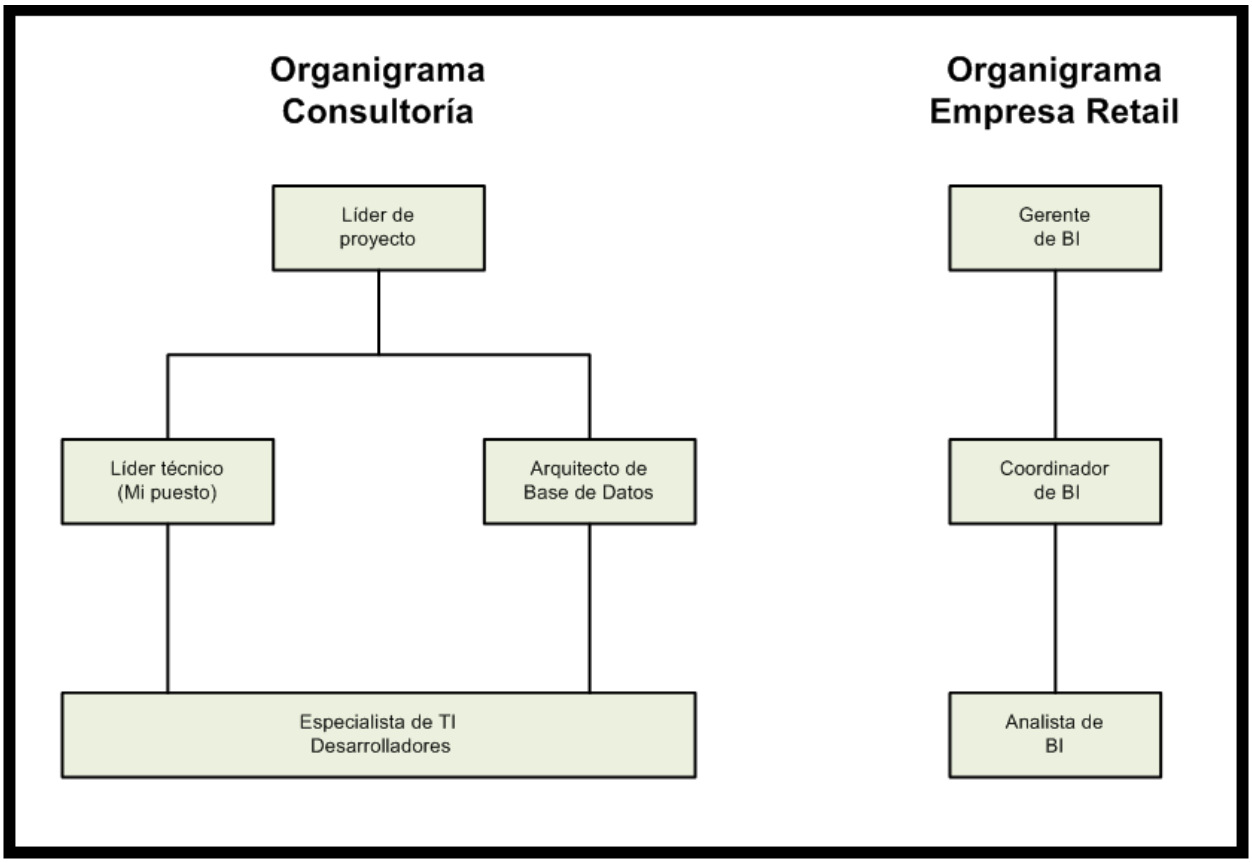


Figura 1. Organigrama del proyecto (Diseño propio)

Debido a mi puesto, realicé en conjunto con el arquitecto de la base de datos el diseño de algunos módulos del DWH, pero aún más importante el diseño de cómo debía de cargarse la información. Para realizar estas actividades, tuve que emplear los conocimientos adquiridos de mi formación como ingeniero en la escuela, así como en proyectos anteriores que hice en mi formación profesional sobre bases de datos y de sistemas operativos en especial de Linux, como se verá en secciones más adelante.

Antecedentes

Debido a los cambios tecnológicos y sociales el comportamiento de compra de los clientes ha cambiado, por lo que las empresas de Retail tienen que cambiar la manera en que lleguen a los sectores de la población. Por lo que el tener un Data Warehouse en un estado óptimo, puede ayudar con análisis muy importantes para cualquier empresa.

Esto lleva al propósito de este documento, que es la integración de un DWH funcional y que pueda ayudar a contestar todas las necesidades que la empresa de Retail tenga, con el objetivo de hacer una buena toma de decisiones.

Para poder ejemplificar el alcance de este proyecto, se tomará como referencia uno de los módulos que diseñé y cómo se hace la carga diaria al DWH. Debido a que este trabajo es sobre manejo de información, la estructura de las tablas mencionadas en este documento, los procesos y estructuras a nivel de sistema operativo, no serán los originales.

El módulo que se empleará para ejemplificar estas actividades será el de *Productos*, escogí este módulo porque además de que yo lo diseñé, los análisis realizados por las áreas usuarias casi siempre incluyen este módulo, esto debido a la interacción que tiene con otros módulos del DWH, en la figura 2 se observa cómo se integra con otros módulos.

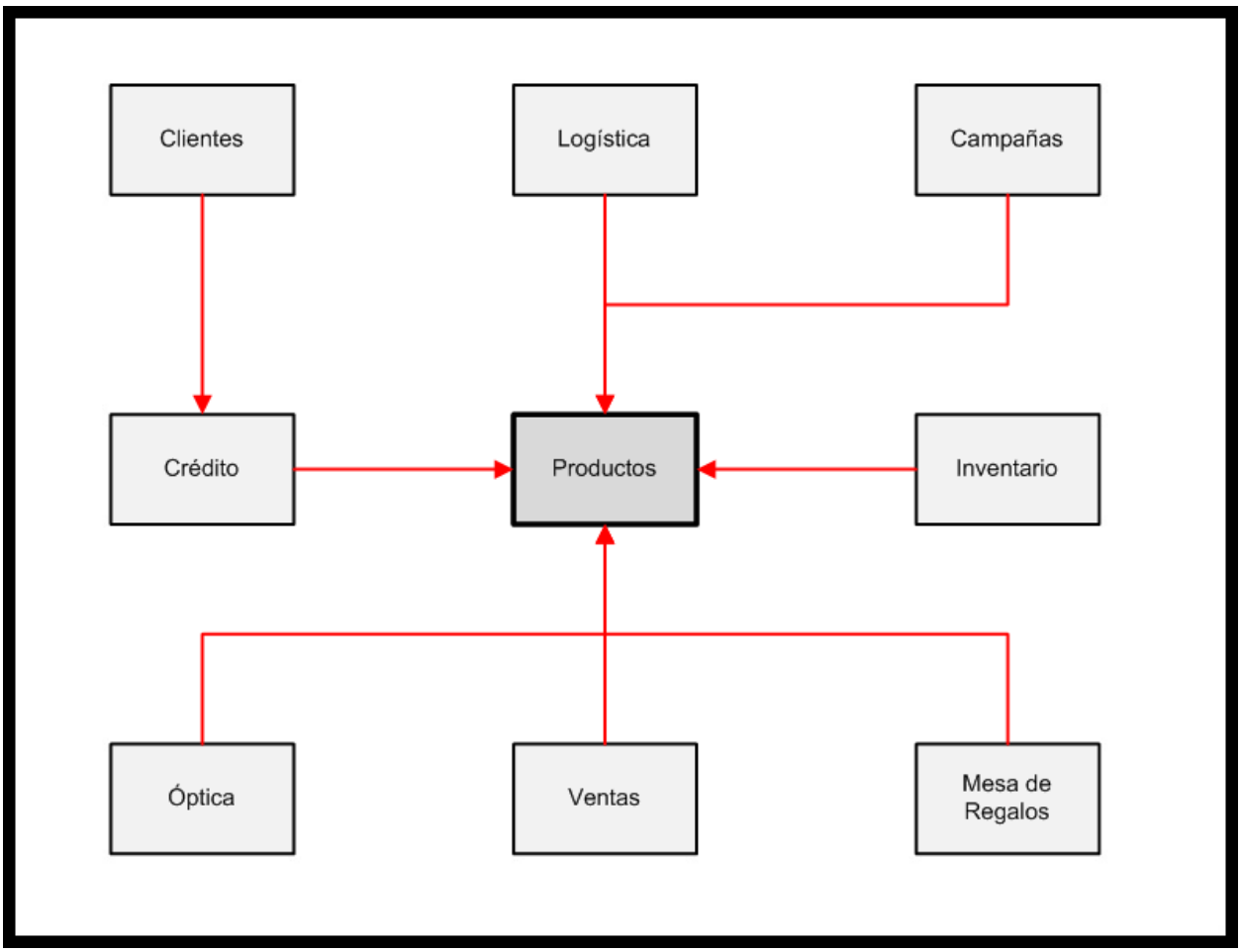


Figura 2. Módulos conectados con el de Productos en el DWH (Diseño propio)

Al realizar este proyecto, además de realizar el DWH tuve que aprender términos de Retail, ya que era necesario al momento de construirlo obtener un mejor funcionamiento del mismo, sobre todo en la integración de los diferentes módulos que conforman el DWH.

Definición del problema

Como se indica en la sección anterior, es necesario la creación de un nuevo DWH, debido a que el anterior ya no respondía de forma adecuada a las necesidades de la empresa, al revisar ese DWH en conjunto con el arquitecto de base de datos, encontramos que había muchas mejoras que se podían realizar.

Uno de los principales problemas que se observaron en el anterior DWH, fueron tablas que tenían información diferente de cuando se habían construido originalmente, por lo que al momento de consultarlas y realizar cruces entre ellas su desempeño ya no era el esperado.

Además, se vieron problemas de formato en algunas columnas lo cual hacía que los cruces tardaran más tiempo, por lo que con la ayuda del área encargada del DWH en la empresa de Retail, al momento de hacer esta revisión se pudo mejorar e integrar algunas tablas en una sola, esto debido a la naturaleza de las mismas.

Problemática

El principal problema que se tuvo al momento de construir este nuevo DWH, fue la restricción que nos impuso el cliente, debido a que varios módulos del anterior Data Warehuose debían de conservarse en su mayoría iguales, esto debido al conocimiento previo que ya se tenía en varias áreas usuarias y la forma en cómo explotaban esa información.

Otro factor con el cual tuve que trabajar fue con la base de datos Teradata, ya que la empresa de Retail ya había realizado la compra para la obtención de la estructura de esta base de datos, por lo que no se pudo aportar un análisis y recomendación de que base de datos era la adecuada para hacer este DWH.

Debido a las circunstancias antes mencionadas, el líder del proyecto, el arquitecto de la base de datos y yo tuvimos varias juntas con las áreas usuarias y los distintos sistemas fuente que alimentan el DWH, ya que era muy importante el saber cuáles eran las necesidades reales y cómo podía ser posible el cubrirlas. Pero el mayor reto fue el lograr que la empresa de Retail entendiera que se debían de hacer algunos cambios a sus procesos, ya que con esto lograría un mejor desempeño y mantenimiento del DWH.

Por último, cómo se menciona en la sección anterior, tuve que aprender varios términos que manejan en la empresa de Retail, ya que tanto el área del DWH como las distintas áreas usuarias tienen un entendimiento a nivel negocio muy alto, por lo que era muy importante el aprender la terminología y poder entender lo que se requería para este nuevo DWH.

Metodología utilizada

La metodología utilizada para este proyecto debido a su naturaleza fue la metodología Agile, esto debido a que es muy común usar esta metodología en proyectos de TI, ya que tiene como principal virtud la flexibilidad y capacidad de modificar el producto a lo largo del proyecto, ya que estas características se van usando al mismo tiempo que se desarrolla.

Esta metodología se basa en dividir el proyecto en fases, conocidas también como sprints, de las cuales el resultado es un producto con una serie de funcionalidades que ya permiten que sea usado. Estas fases suceden hasta que se consigue el total de la funcionalidad del proyecto y se componen de la siguiente manera

- Inicio. Se escoge del total de los objetivos del proyecto aquellos que deben ser implementados en el sprint, para generar un producto funcional. Con base en estos objetivos se define la duración del sprint (entre una semana y un mes), además de las tareas que lo componen.

- Desarrollo del sprint. Primero se planifican y después se ejecutan las tareas, las cuales deben de supervisarse en reuniones diarias donde se ven las tareas ejecutadas, las que siguen en curso y las pendientes. Además, en estas juntas se revisan los posibles impedimentos y/o restricciones de cada tarea.
- Cierre. Al final de cada sprint se revisa que se hayan completado todas las tareas y objetivos definidos al inicio, esto se hace con una presentación del producto funcional.

Con la posibilidad de tener un producto funcional y utilizable al final de cada sprint, permite ajustar los objetivos del proyecto, por lo que esto ayuda a tener un producto final que cumpla con las expectativas del usuario.

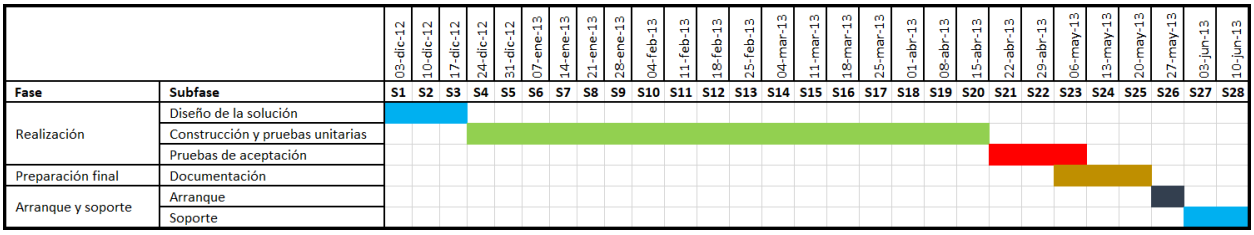
Hay que tener mucho cuidado cuando se menciona que un producto es funcional, ya que esto no significa que el proyecto esté terminado ni mucho menos que ya pueda implementarse en un ambiente productivo, esta parte se hace en el último sprint cuando ya no hay objetivos que cumplir en el proyecto. En la figura 3 se observa cómo funciona esta metodología.

Figura 3. Metodología Agile (https://openwebinars.net/blog/que-es-la-metodologia-agile/)

Para este proyecto realicé las siguientes actividades, esto independientemente de las fases de la metodología antes mencionada

- Obtención de los nuevos datos
- Arquitectura del DWH
- Construcción de los procesos de extracción, transformación y carga (ETL) del DWH y pruebas unitarias
- Pruebas integrales
- Puesta en producción y soporte

El tiempo en cómo se dividió este proyecto originalmente fue en un poco más de 6 meses, teniendo el siguiente cronograma de actividades descrito en la tabla 1.



		03-dic-12	10-dic-12	17-dic-12	24-dic-12	31-dic-12	07-ene-13	14-ene-13	21-ene-13	28-ene-13	04-feb-13	11-feb-13	18-feb-13	25-feb-13	04-mar-13	11-mar-13	18-mar-13	25-mar-13	01-abr-13	08-abr-13	15-abr-13	22-abr-13	29-abr-13	06-may-13	13-may-13	20-may-13	27-may-13	03-jun-13	10-jun-13
Fase	**Subfase**	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**	**S11**	**S12**	**S13**	**S14**	**S15**	**S16**	**S17**	**S18**	**S19**	**S20**	**S21**	**S22**	**S23**	**S24**	**S25**	**S26**	**S27**	**S28**
Realización	Diseño de la solución																												
	Construcción y pruebas unitarias																												
	Pruebas de aceptación																												
Preparación final	Documentación																												
Arranque y soporte	Arranque																												
	Soporte																												

Tabla 1. Cronograma original de actividades (Diseño propio)

A continuación, describo cada una de las actividades de forma detallada, así como la importancia de cada una y los problemas que se tuvieron con este proyecto, para obtener una solución que cumpliera con las expectativas del cliente.

Obtención de los nuevos datos

Para esta etapa dentro del proyecto, fui a varias juntas con la gente que maneja la información de la empresa, ya que ellos podían indicarme cómo era el envío de los datos para su posterior carga en el DWH actual. Además, en conjunto con el arquitecto de la base de datos les expliqué cuál era la necesidad del negocio, para la obtención de los nuevos datos y poder contestar esas necesidades.

Esta actividad no se tenía contemplada en el plan original del proyecto, ya que en el contrato original se especificó que la gente de la empresa de Retail nos proporcionaría todos los insumos necesarios para comenzar a desarrollar. Sin embargo, al momento de tener la junta inicial donde presentamos el cronograma de actividades, así como la forma en que el equipo de desarrollo iba a trabajar, se nos indicó que debíamos de ayudar con la obtención de los datos de los sistemas fuente que alimentarían al DWH.

Después de las juntas, esta actividad se dividió en dos partes, la primera consistió en realizar la documentación para pedir la información que alimentaría al DWH, estos documentos son un estándar que tiene la empresa de Retail para este tipo de requerimientos, por lo que se tuvo que trabajar en conjunto con el equipo de la empresa de Retail.

La segunda parte consistió en el proceso interno de la empresa de Retail para obtener los datos, de acuerdo con lo establecido en los documentos. En esta etapa, el área del DWH de la empresa de Retail se encargó de gestionar todos los procesos, de nuestra parte nos encargamos de validar que los datos llegaran como se especificaron y poder hacer la retroalimentación en caso de encontrar algún error en los datos.

Debido a la cantidad de procesos que se hicieron para este proyecto, esta actividad tardó aproximadamente 2 meses, también influyó el hecho que no todos los sistemas fuentes pudieron entregar sus procesos al mismo tiempo. Por lo que al momento de tener toda la documentación terminada y sin tener datos que validar, comencé con el diseño en conjunto con el arquitecto de la base de datos.

Lo primero que hicimos fue el análisis de cómo el DWH interactúa con las diferentes áreas usuarias, por lo que después de ver de forma general la arquitectura del DWH las actividades se dividieron, ya que el arquitecto de la base de datos comenzó a diseñar el modelo del DWH y yo comencé a diseñar el proceso de carga de los datos. Estas actividades no se tenían contempladas hacerlas de esta forma, pero debido al tiempo transcurrido con la actividad anterior, fue una de las mejores opciones que pensamos que nos ayudaría a minimizar el tiempo perdido. A continuación, se describe a detalle cómo es la arquitectura y posteriormente el proceso de carga.

Arquitectura del DWH

La arquitectura a nivel general del DWH está constituida, mediante procesos batch[2] que los sistemas fuente tienen, para enviar la información en forma de interfaces a un servidor llamado servidor DataStage[3], una vez que llegan las interfaces se realiza la carga en el DWH mediante procesos ETL, como se muestra en la figura 4, poniendo de una forma muy simple como interactúa el DWH, desde los sistemas fuentes hasta los usuarios que explotan dicha información.

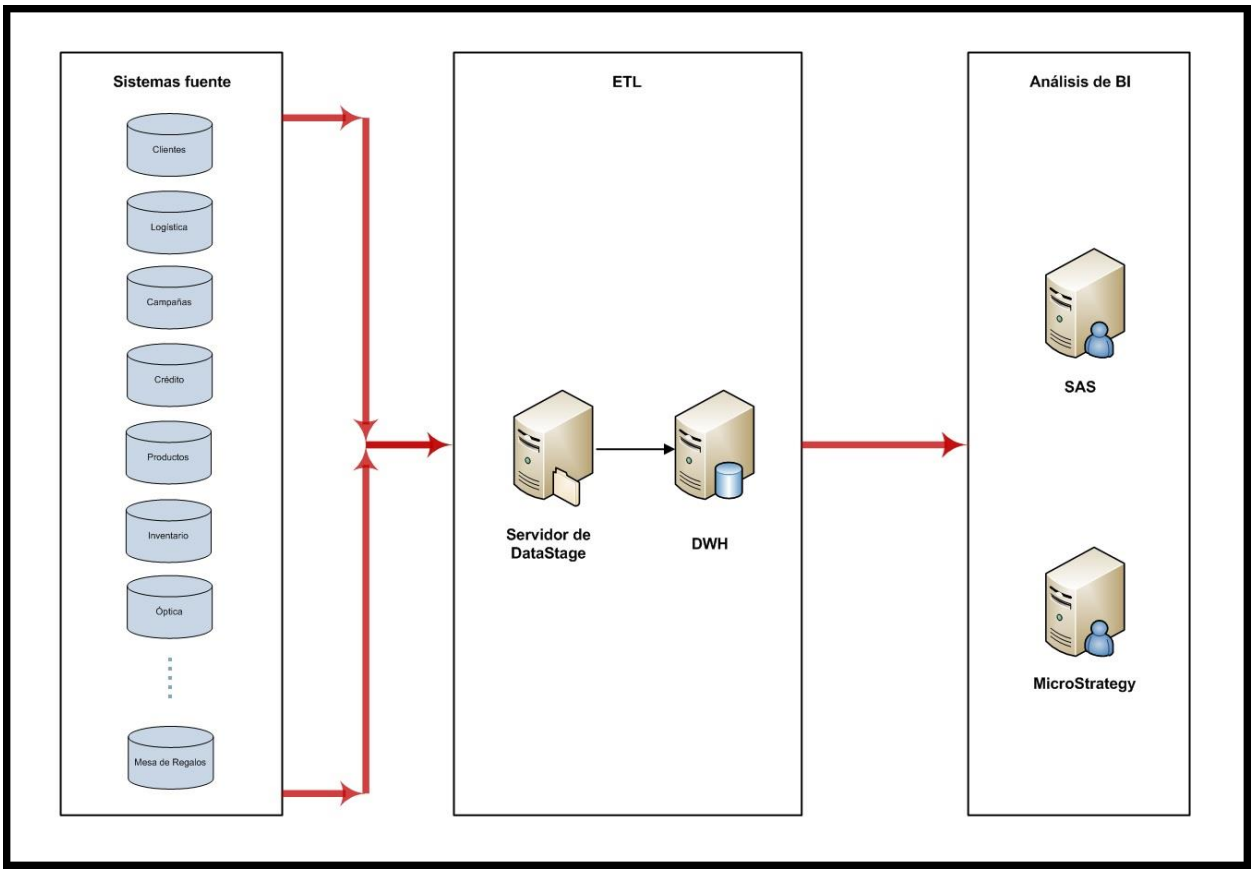


Figura 4. Arquitectura general del DWH (Diseño propio)

[2] Proceso que es ejecutado sin el control o supervisión directa del usuario. Ver anexo

[3] Es una herramienta ETL para integración de datos. Ver anexo

El DWH se compone de los siguientes programas que son usados para su carga, uso y explotación

- Un servidor SUSE Linux Enterprise Server 11 (x86_64), donde los sistemas fuentes depositan las interfaces que deben de ser cargadas en el proceso batch diario, conocido como servidor de DataStage
- Una herramienta ETL IBM InfoSphere DataStage 11.5 instalada en el servidor SUSE Linux
- La base de datos Teradata 13.10
- Para hacer minería de datos SAS 9.4
- Para reporteo de datos MicroStrategy 11.1

Por lo que para cargar los archivos se hace uso de varias tecnologías, un sistema operativo Linux, una herramienta ETL y scripts propios de Teradata los cuales se describen a continuación.

Construcción de los procesos ETL del DWH y pruebas unitarias

Para realizar la carga de los procesos del DWH en cada una de sus etapas, al igual que los procesos de los sistemas fuente, se crearon procesos de tipo batch, los cuales son controlados mediante una herramienta llamada Control–M. Además, la carga al DWH se hace por etapas, como se observa en la figura 5.

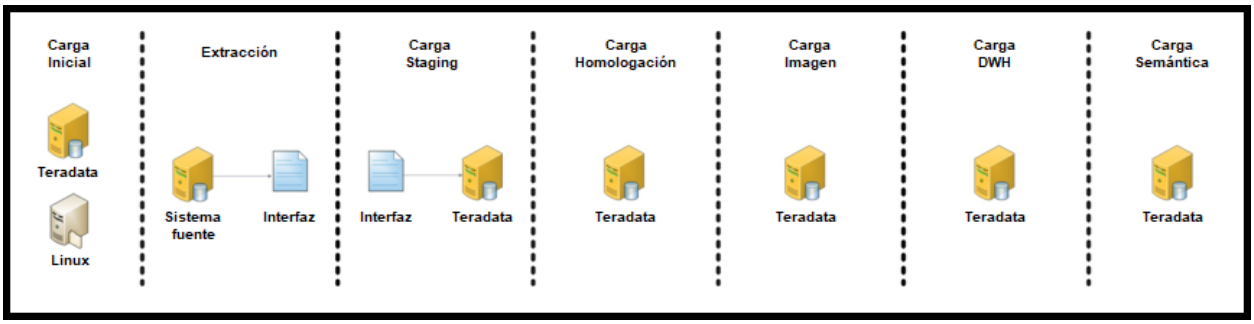


Figura 5. Etapas de carga en el DWH (Diseño propio)

Este diseño es debido a la cantidad de procesos que se tienen, por lo que el ETL se compone de las siguientes etapas

- Carga inicial del DWH
- Extracción de interfaces de los sistemas fuente
- Carga a las tablas de Staging
- Carga a las tablas de Homologación, Imagen y DWH
- Carga a las tablas de Semántica

Las etapas de análisis de BI son una consecuencia de la carga de información de un DWH, por lo que sólo se mencionan, ya que no son parte de este proyecto. A continuación, describo cada una de las fases de carga y el por qué diseñé de esta forma este proceso.

Carga inicial del DWH

Esta etapa es la más importante, ya que cuando hice el diseño inicial del DWH se requería una forma simple de poder cargar los archivos generados por los sistemas fuentes y el poder tener dichos archivos por un tiempo como respaldo, esto se logró gracias a la metodología usada en el proyecto, ya que en cada sprint de esta etapa el cliente nos dio mucha retroalimentación. Por lo que además de diseñar el proceso de carga al DWH, tuve que analizar la forma en que podría hacerse dicha carga de una forma simple.

Lo primero que diseñé fue la forma en cómo se debía de crear la estructura a nivel sistema operativo, ya que las primeras etapas serían el manipular las interfaces que los sistemas fuentes nos enviarían al servidor de DataStage, descrito en la sección anterior. La gente de la empresa de Retail nos dio un file system[4], en el cual hice la estructura de cómo debía de quedar en el servidor de DataStage, para el manejo de todos los archivos que son necesarios para la carga diaria al DWH.

[4] Estructura la información guardada en una unidad de almacenamiento. Ver anexo

El file system se compone de la siguiente manera, se tiene una ruta origen la cual fue definida por la gente de la empresa de Retail y a partir de dicha ruta hice la estructura para que los procesos del DWH funcionaran. La ruta es la siguiente */datastage_data/empresa_A/DWH/* tal como se observa en la figura 6.

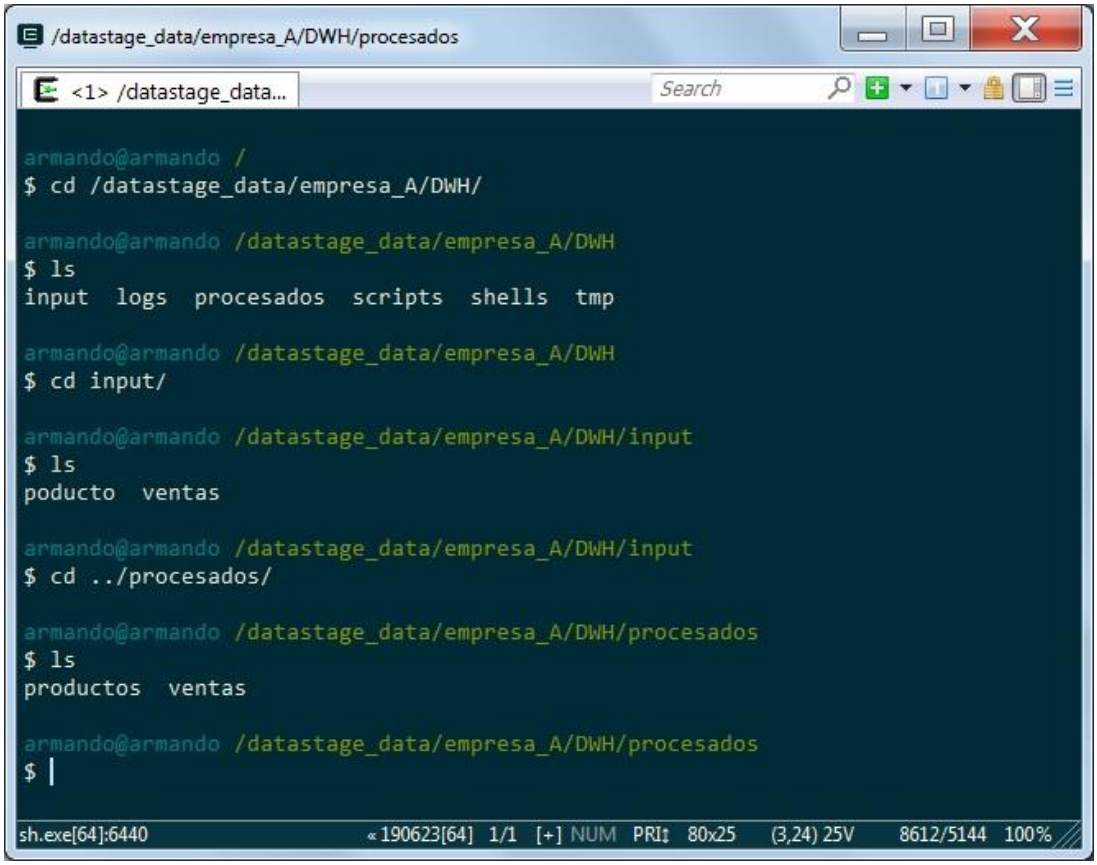


Figura 6. File system del servidor de DataStage (Emulador Cygwin)

Debido a que la ruta original muestra el nombre de la empresa de Retail, se creó de manera local en un emulador de Linux la estructura, pero cambiando el nombre original de la empresa por empresa_A.

Los directorios que agregué a partir de esa ruta, fueron uno de logs para poder tener un control sobre la carga del DWH, uno de scripts la cual se describirá en las etapas de carga, uno de shells el cual sirve para tener todos los archivos ejecutables que se

necesitan para hacer el proceso de carga de tipo batch, uno de input donde además, se crearon subdirectorios para cada sistema fuente y ahí se depositan las interfaces generadas por cada sistema fuente, en este caso sólo agregué los de productos y las ventas para tener el ejemplo de cómo está constituido el file system, también uno de procesados con dos subdirectorios, los cuales sirven para tener de respaldo las interfaces que ya han sido procesadas, por último un directorio de archivos temporales.

Una vez que se tenía la estructura a nivel sistema operativo, comencé a diseñar la forma en cómo se debían de ejecutar los procesos por batch, siempre consideré la forma en que Teradata debía gestionar los logs de cada proceso. Una ventaja que observé en Teradata es que está diseñada con una arquitectura paralela, logrando gestionar de forma eficaz requerimientos complejos, simplificando la administración y gestión del DWH. Esto se logra distribuyendo los datos y balanceando la carga de trabajo de forma automática, como se muestra en la figura 7.

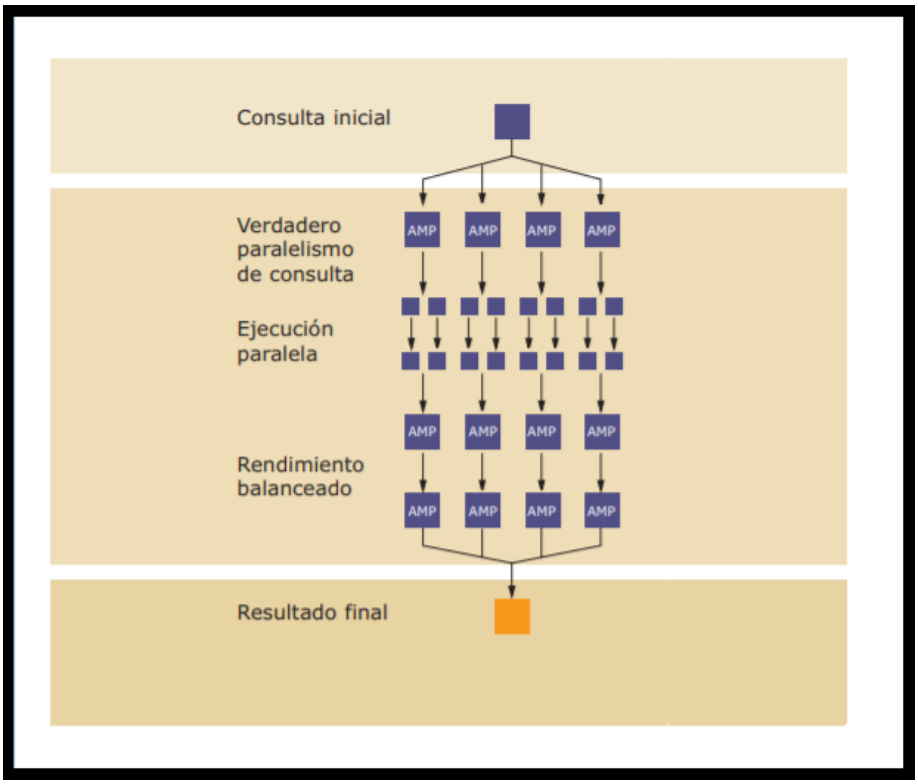


Figura 7. Arquitectura de Teradata (Base de Datos Teradata: Introducción Técnica.pdf)

Como se observa en la figura anterior, aún cuando Teradata es un gestor de base de datos relacional, está diseñado para soportar paralelismo, ya que su arquitectura le permite descomponer consultas complejas, entre múltiples unidades de trabajo paralelas en el software de la base de datos, dichas unidades son denominadas AMP (Procesadores de módulos de acceso).

A cada AMP le corresponde una parte del espacio y de los datos en la base de datos, teniendo varios AMPs Teradata no está condicionada por la plataforma de hardware para soportar paralelismo, escalabilidad o alta disponibilidad. Por lo que estas características son inherentes a su propia arquitectura de software, sin importar el sistema operativo o su configuración.

Teniendo en cuenta todos los elementos anteriores para el proceso de carga al DWH, el proceso de carga inicial lo diseñé como se muestra en la figura 8.

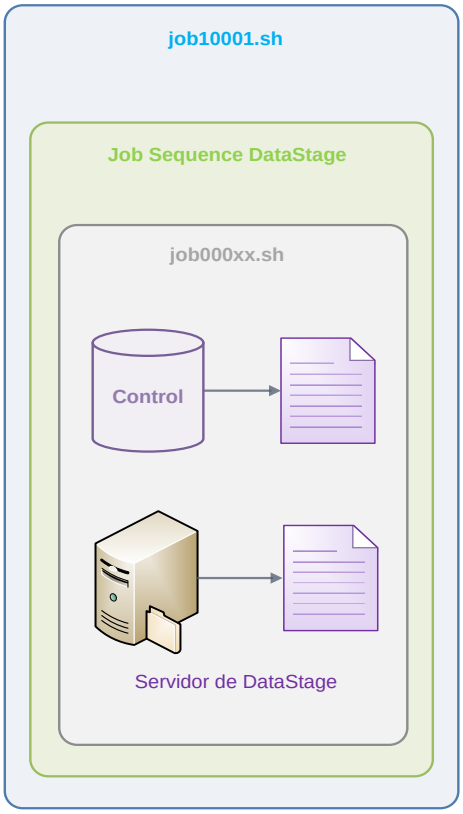


Figura 8. Proceso de Carga Inicial (Diseño propio)

Al diseñar este proceso, tuve que considerar algunos factores que fueron pedidos por la empresa de Retail y otros por mi equipo, por lo que tuve que crear una etapa de carga inicial en el DWH.

Este proceso de carga inicial prepara la ejecución de carga de todos los procesos del DWH, además gracias a que uno de los componentes de software es un servidor Linux, para ejecutar cada proceso diseñé archivos de tipo shell, los cuales son descritos en la tabla 2.

Número de shell	Tipo de shell
job00000.sh	Shell de carga a tablas o procesos de control
job01000.sh	Shell de extracción de sistema fuente
job02000.sh	Shell de carga a tablas de staging
job03000.sh	Shell de carga a tablas de homologación
job04000.sh	Shell de carga a tablas de imagen (DWI)
job05000.sh	Shell de carga a tablas de DWH
job06000.sh	Shell de carga a tablas de semántica
job10000.sh	Shell de proceso en Control – M

Tabla 2. Tipos de shell para las etapas de carga del DWH (Diseño propio)

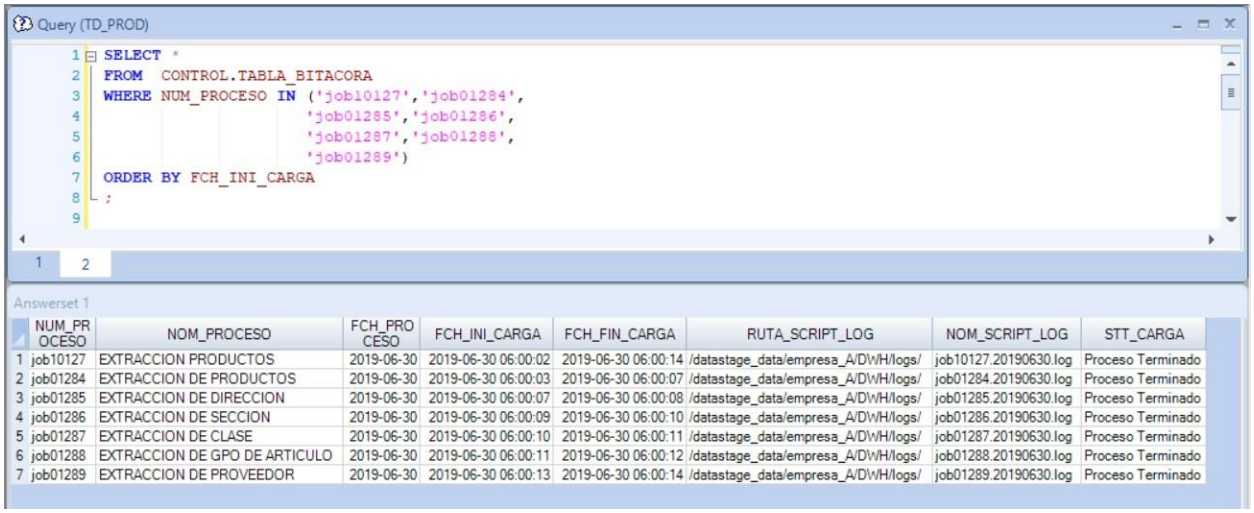
Para el proceso de carga inicial se crearon varios shells con la numeración job0000x.sh para hacer las siguientes acciones

- Actualización de la tabla de bitácora de procesos
- Creación de un archivo de control de usuarios
- Creación de un archivo de control de procesos
- Depuración de interfaces y logs de carga
- Reinicio de los Jobs de DataStage

A continuación, describo cada una de las acciones que realiza la carga inicial a detalle, donde se describe la importancia de esta etapa, para el correcto funcionamiento de la carga diaria del DWH.

Actualización de la tabla de bitácora de procesos

Para tener una manera rápida de validar los procesos de carga creé una tabla de bitácora, para poder revisar de forma rápida el estatus de cada uno de los procesos que se ejecuta para la carga del DWH, como se muestra en la figura 9.



	NUM_PROCESO	NOM_PROCESO	FCH_PROCESO	FCH_INI_CARGA	FCH_FIN_CARGA	RUTA_SCRIPT_LOG	NOM_SCRIPT_LOG	STT_CARGA
1	job10127	EXTRACCION PRODUCTOS	2019-06-30	2019-06-30 06:00:02	2019-06-30 06:00:14	/datastage_data/empresa_A/DWH/logs/	job10127.20190630.log	Proceso Terminado
2	job01284	EXTRACCION DE PRODUCTOS	2019-06-30	2019-06-30 06:00:03	2019-06-30 06:00:07	/datastage_data/empresa_A/DWH/logs/	job01284.20190630.log	Proceso Terminado
3	job01285	EXTRACCION DE DIRECCION	2019-06-30	2019-06-30 06:00:07	2019-06-30 06:00:08	/datastage_data/empresa_A/DWH/logs/	job01285.20190630.log	Proceso Terminado
4	job01286	EXTRACCION DE SECCION	2019-06-30	2019-06-30 06:00:09	2019-06-30 06:00:10	/datastage_data/empresa_A/DWH/logs/	job01286.20190630.log	Proceso Terminado
5	job01287	EXTRACCION DE CLASE	2019-06-30	2019-06-30 06:00:10	2019-06-30 06:00:11	/datastage_data/empresa_A/DWH/logs/	job01287.20190630.log	Proceso Terminado
6	job01288	EXTRACCION DE GPO DE ARTICULO	2019-06-30	2019-06-30 06:00:11	2019-06-30 06:00:12	/datastage_data/empresa_A/DWH/logs/	job01288.20190630.log	Proceso Terminado
7	job01289	EXTRACCION DE PROVEEDOR	2019-06-30	2019-06-30 06:00:13	2019-06-30 06:00:14	/datastage_data/empresa_A/DWH/logs/	job01289.20190630.log	Proceso Terminado

Figura 9. Tabla de bitácora de procesos del DWH (Asistente de Teradata)

Esta tabla tiene la información del proceso que se ejecutó, la fecha de cuando fue ejecutado, la fecha y hora de inicio, la fecha y hora de fin, la ruta donde se puede ver su archivo de log, el nombre del archivo del log y el estatus final del proceso.

Debido a las bondades de Teradata en cuanto a su arquitectura y que no es necesario crear DB Links[5], ya que sólo se debe de otorgar permisos de lectura entre bases de datos, cuando se requiere hacer consultas entre distintas bases, lo mismo sucede con los usuarios, sólo se le dan permisos de lectura y/o escritura a las bases de datos que necesita leer o manipular, fue posible la creación de varias bases de datos por cada etapa del DWH, las cuales se muestran en la figura 10.

[5] Es un objeto que permite una referencia a una tabla o vista a una base de datos remota. Ver anexo

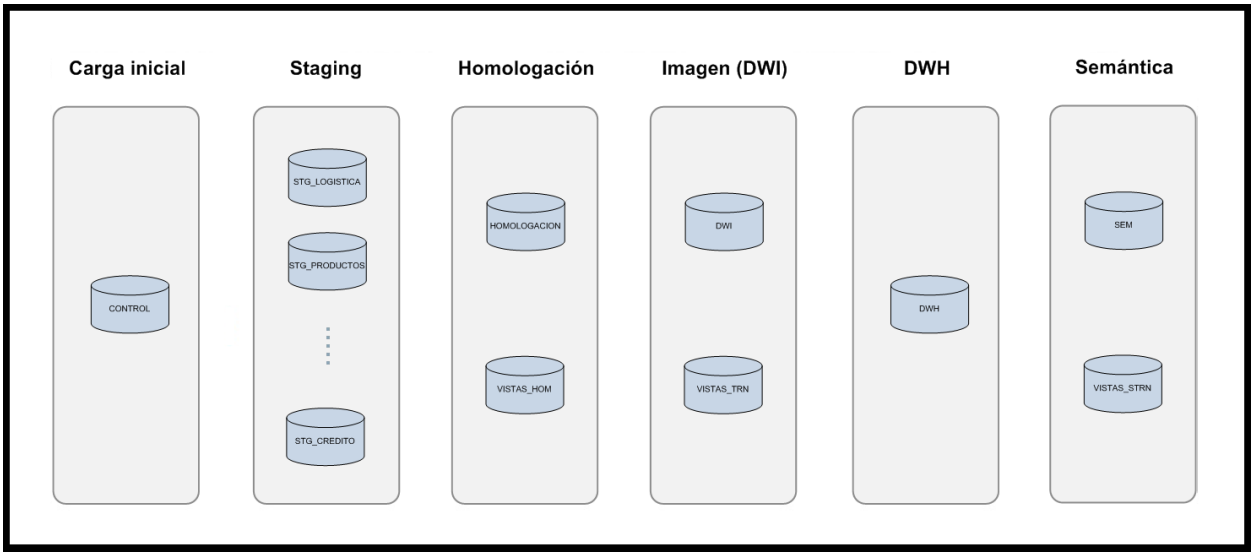
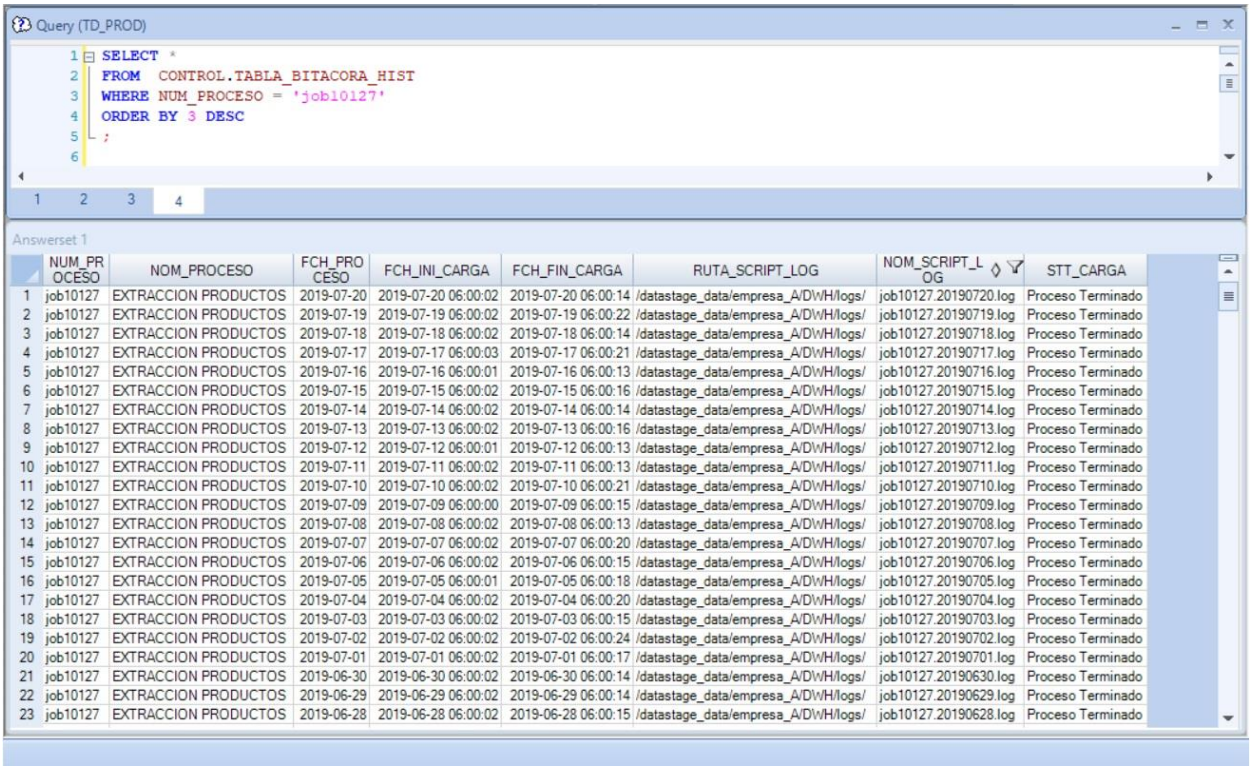


Figura 10. Bases de datos del DWH (Diseño propio)

Considerando estas bondades que otorga Teradata fue posible la creación de esta tabla en una base de datos de control, la cual todos los usuarios creados para el proceso de carga al DWH pueden acceder y cargar el estado de sus respectivos procesos.

Existen dos tablas de bitácora, la del proceso batch que se ejecuta diario y una histórica que tiene todos los procesos que se han ejecutado desde el inicio del DWH hasta la ejecución del día anterior. Ambas tablas tienen la misma estructura de datos mostrada en la figura 9.

Este proceso se hace en dos partes, el primero es actualizar la tabla histórica con la información de la tabla de bitácora, esto para tener siempre la evidencia de cómo fue la ejecución de todos los procesos, tal como se muestra en la figura 11. Esta actualización se hace mediante un script llamado bteq, el cual tiene código SQL y es ejecutado con un comando en un shell.



	NUM_PROCESO	NOM_PROCESO	FCH_PROCESO	FCH_INI_CARGA	FCH_FIN_CARGA	RUTA_SCRIPT_LOG	NOM_SCRIPT_LOG	STT_CARGA
1	job10127	EXTRACCION PRODUCTOS	2019-07-20	2019-07-20 06:00:02	2019-07-20 06:00:14	/datastage_data/empresa_A/DWH/logs/	job10127.20190720.log	Proceso Terminado
2	job10127	EXTRACCION PRODUCTOS	2019-07-19	2019-07-19 06:00:02	2019-07-19 06:00:22	/datastage_data/empresa_A/DWH/logs/	job10127.20190719.log	Proceso Terminado
3	job10127	EXTRACCION PRODUCTOS	2019-07-18	2019-07-18 06:00:02	2019-07-18 06:00:14	/datastage_data/empresa_A/DWH/logs/	job10127.20190718.log	Proceso Terminado
4	job10127	EXTRACCION PRODUCTOS	2019-07-17	2019-07-17 06:00:03	2019-07-17 06:00:21	/datastage_data/empresa_A/DWH/logs/	job10127.20190717.log	Proceso Terminado
5	job10127	EXTRACCION PRODUCTOS	2019-07-16	2019-07-16 06:00:01	2019-07-16 06:00:13	/datastage_data/empresa_A/DWH/logs/	job10127.20190716.log	Proceso Terminado
6	job10127	EXTRACCION PRODUCTOS	2019-07-15	2019-07-15 06:00:02	2019-07-15 06:00:16	/datastage_data/empresa_A/DWH/logs/	job10127.20190715.log	Proceso Terminado
7	job10127	EXTRACCION PRODUCTOS	2019-07-14	2019-07-14 06:00:02	2019-07-14 06:00:14	/datastage_data/empresa_A/DWH/logs/	job10127.20190714.log	Proceso Terminado
8	job10127	EXTRACCION PRODUCTOS	2019-07-13	2019-07-13 06:00:02	2019-07-13 06:00:16	/datastage_data/empresa_A/DWH/logs/	job10127.20190713.log	Proceso Terminado
9	job10127	EXTRACCION PRODUCTOS	2019-07-12	2019-07-12 06:00:01	2019-07-12 06:00:13	/datastage_data/empresa_A/DWH/logs/	job10127.20190712.log	Proceso Terminado
10	job10127	EXTRACCION PRODUCTOS	2019-07-11	2019-07-11 06:00:02	2019-07-11 06:00:13	/datastage_data/empresa_A/DWH/logs/	job10127.20190711.log	Proceso Terminado
11	job10127	EXTRACCION PRODUCTOS	2019-07-10	2019-07-10 06:00:02	2019-07-10 06:00:21	/datastage_data/empresa_A/DWH/logs/	job10127.20190710.log	Proceso Terminado
12	job10127	EXTRACCION PRODUCTOS	2019-07-09	2019-07-09 06:00:00	2019-07-09 06:00:15	/datastage_data/empresa_A/DWH/logs/	job10127.20190709.log	Proceso Terminado
13	job10127	EXTRACCION PRODUCTOS	2019-07-08	2019-07-08 06:00:02	2019-07-08 06:00:13	/datastage_data/empresa_A/DWH/logs/	job10127.20190708.log	Proceso Terminado
14	job10127	EXTRACCION PRODUCTOS	2019-07-07	2019-07-07 06:00:02	2019-07-07 06:00:20	/datastage_data/empresa_A/DWH/logs/	job10127.20190707.log	Proceso Terminado
15	job10127	EXTRACCION PRODUCTOS	2019-07-06	2019-07-06 06:00:02	2019-07-06 06:00:15	/datastage_data/empresa_A/DWH/logs/	job10127.20190706.log	Proceso Terminado
16	job10127	EXTRACCION PRODUCTOS	2019-07-05	2019-07-05 06:00:01	2019-07-05 06:00:18	/datastage_data/empresa_A/DWH/logs/	job10127.20190705.log	Proceso Terminado
17	job10127	EXTRACCION PRODUCTOS	2019-07-04	2019-07-04 06:00:02	2019-07-04 06:00:20	/datastage_data/empresa_A/DWH/logs/	job10127.20190704.log	Proceso Terminado
18	job10127	EXTRACCION PRODUCTOS	2019-07-03	2019-07-03 06:00:02	2019-07-03 06:00:15	/datastage_data/empresa_A/DWH/logs/	job10127.20190703.log	Proceso Terminado
19	job10127	EXTRACCION PRODUCTOS	2019-07-02	2019-07-02 06:00:02	2019-07-02 06:00:24	/datastage_data/empresa_A/DWH/logs/	job10127.20190702.log	Proceso Terminado
20	job10127	EXTRACCION PRODUCTOS	2019-07-01	2019-07-01 06:00:02	2019-07-01 06:00:17	/datastage_data/empresa_A/DWH/logs/	job10127.20190701.log	Proceso Terminado
21	job10127	EXTRACCION PRODUCTOS	2019-06-30	2019-06-30 06:00:02	2019-06-30 06:00:14	/datastage_data/empresa_A/DWH/logs/	job10127.20190630.log	Proceso Terminado
22	job10127	EXTRACCION PRODUCTOS	2019-06-29	2019-06-29 06:00:02	2019-06-29 06:00:14	/datastage_data/empresa_A/DWH/logs/	job10127.20190629.log	Proceso Terminado
23	job10127	EXTRACCION PRODUCTOS	2019-06-28	2019-06-28 06:00:02	2019-06-28 06:00:15	/datastage_data/empresa_A/DWH/logs/	job10127.20190628.log	Proceso Terminado

Figura 11. Tabla de bitácora histórica del DWH (Asistente de Teradata)

Estos scripts son una abreviatura de Basic Teradata Query, que es un programa basado en comandos de uso general que permite a los usuarios comunicarse con uno o más sistemas de base de datos Teradata. Usando un bteq se puede enviar consultas SQL a la base de datos Teradata. Estos scripts pueden ser tan simples o complejos de acuerdo con la necesidad, por lo que se puede meter comandos de tipo DDL o Leguaje de Definición de Datos

- CREATE
- DROP
- ALTER

También pueden usarse comandos de tipo DML o Lenguaje de Manipulación de Datos

- SELECT
- INSERT
- UPDATE

- DELETE

Por lo que se tiene una gran versatilidad al momento de crear estos scripts, cabe mencionar que estos scripts son archivos físicos que se encuentran dentro del servidor de DataStage en la ruta de scripts. El código del bteq es el siguiente.

```
.SET ERROROUT STDOUT;
.LOGON ${var_carga}
.SET WIDTH 1000;
.IF errorlevel <> 0 THEN .QUIT 20


-- Limpia datos antes de realizar carga --
DELETE FROM ${NOM_DB}.${NOM_TAB}_HIST
WHERE FCH_PROCESO = Cast('${FH_PROC}' AS DATE Format 'YYYYMMDD');


-- Inserta a historica ---
INSERT INTO ${NOM_DB}.${NOM_TAB}_HIST
SELECT
              NUM_PROCESO
             ,NOM_PROCESO
             ,FCH_PROCESO
             ,FCH_INI_CARGA
             ,FCH_FIN_CARGA
             ,RUTA_SCRIPT_LOG
             ,NOM_SCRIPT_LOG
             ,STT_CARGA
FROM ${NOM_DB}.${NOM_TAB};
```

En donde las primeras líneas son parámetros que sirven para ejecutar el código SQL, de las cuales la segunda línea es el comando con el que se conecta a la base de datos, las demás líneas son las que hacen la carga de la tabla de bitácora hacia la tabla histórica.

Además, para evitar que cualquier persona pueda ver datos sensibles, el nombre de la tabla, la base de datos y la conexión se obtienen por medio de parámetros los cuales se identifican porque empiezan con el signo de $ y están entre corchetes, que se le pasan por medio del shell al bteq.

Este proceso es ejecutado por un shell de control el job00004.sh, el cual genera un log al momento de ser ejecutado, esto para poder ver de una forma más simple cuál fue el error en caso de que el proceso tenga una falla. Parte del código del shell es el siguiente.

```
echo 'Comienza ejecucion' `date`

FH_PROC=`date "+%Y%m%d"`; export FH_PROC
NOM_TB=TABLA_BITACORA; export NOM_TB
NOM_DB=CONTROL; export NOM_DB
NU_JOB = job00004

echo "_____________________________________________________________________________"
echo "                                                                             "
echo "  Realizando proceso de Historia de Bitacora $NOM_TAB    ...                 "
echo "_____________________________________________________________________________"

# Inserta o actualiza el proceso de carga comienza en la tabla de bitacora
disp_BTQ_sp "$NUM_JOB, '$NU_JOB', '$INF_FASE', NULL, 1, '$HNOM_TB',
'$FECHA_PROCESO', 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, '$RUTA_LOG',
NULL" "ini_his"

var_carga=${INF_HOST}/${INF_USR},${INF_PASS}; export var_carga
var_btq=${PATH_SCRIPTS}./${SCRIPT_HIS}
var_log=${PATH_LOGS}${NU_JOB}_hisbit.${FECHA}.log

ksh  $var_btq > $var_log

salida_bteq=`grep "(return code)" $var_log |cut -d" " -f7 | head -1`

if [ "{$salida_bteq}" != "{0}" ]
      then
            echo "*--- Error en proceso bteq de ejecucion historica  ---*"
            echo "Código de error: $salida_bteq"

            # Actualiza el error del proceso de carga en la tabla de bitacora
            disp_BTQ_sp "$NUM_JOB,'$NU_JOB', '$INF_FASE', NULL, 1, 'H$NOM_TB',
'$FECHA_PROCESO', 3, 0, 0, 0, 0, 0,'${SCRIPT_HIS}',
'${NU_JOB}_hisbit.${FECHA}.log', '$RUTA_LOG', NULL" "err"

            echo "\n ========[   Termino proceso de historica.....  ]========"
            echo "`date '+%a %Y-%b-%d %H:%M:%S'`"
            exit 1
      else
            echo "*--- Carga de Historica exitosa  ---*"

            RECHAZADOS=0
            INSERTADOS=`grep "Insert completed" $var_log |cut -d" " -f5`

            if  [ "$INSERTADOS" = "No" ];then
                  INSERTADOS=0
            elif  [ "$INSERTADOS" = "One" ];then
                  INSERTADOS=1
```

```
            fi

            # Actualiza el estatus de fin de carga en la tabla de bitacora
            disp_BTQ_sp "$NUM_JOB,'$NU_JOB', '$INF_FASE', NULL, 1, 'H$NOM_TB',
'$FECHA_PROCESO', 2, $INSERTADOS, $INSERTADOS, 0, 0, 0, '${SCRIPT_HIS}',
'${NU_JOB}_hisbit.${FECHA}.log', '$RUTA_LOG', NULL" "fin_his"

            rm -f $var_btq
      fi

echo "_____________________________________________________________________________"
echo "                                                                             "
echo 'Finaliza ejecucion' `date`
exit 0
```

Este código lo que hace es primero obtiene la fecha de proceso del servidor, así como la tabla y la base de datos que se van a ocupar, en este caso los datos de la tabla de bitácora, después inserta o actualiza el registro en la tabla de bitácora por medio de la función disp_BTQ_sp, dicha función fue realizada por uno de los desarrolladores con base en las especificaciones que diseñé, esta acción siempre se hace en todos los procesos de carga del DWH.

Una vez que se insertó o actualizó el registro en la tabla de bitácora se cargan las variables necesarias para ejecutar el insert en la tabla de bitácora histórica, teniendo las siguientes variables:

- var_carga. Contiene el host, usuario y contraseña de la base datos
- var_btq. Contiene el nombre del script que se va a ejecutar
- var_log. Contiene el nombre del log generado al ejecutar el proceso de carga

La ejecución se realiza mediante la línea *ksh $var_btq > $var_log*. Por último, se analiza la ejecución de la carga y se actualiza el resultado en la tabla de bitácora, ya sea con error o con una ejecución exitosa, si termina con error el código de salida del shell es 1 y en caso de ser exitoso el código de salida del shell es 0.

Los logs mencionados anteriormente son los siguientes, uno es para el proceso general el cual es lo que se va generando con la ejecución que va realizando el shell y el otro

es la ejecución que va realizando la carga a la tabla de bitácora. A continuación, pongo el log del proceso general.

```
Comienza ejecucion Sun Jul 7 01:27:38 CDT 2019
_____________________________________________________________________

  Proceso de Ejecución para JOB: job00004
_____________________________________________________________________

Comienza ejecucion Sun Jul 7 01:27:38 CDT 2019

_____________________________________________________________________

  Realizando proceso de Historia de Bitacora    ...
_____________________________________________________________________

_____________________________________________________________________

    Registrando ejecución en bitacora ini_his ...
_____________________________________________________________________

*--- Finaliza Registro de ejecución en Bitacora ini_his correcta  ---*
*--- Carga de Historica exitosa  ---*

_____________________________________________________________________

    Registrando ejecucion en bitacora fin_his ...
_____________________________________________________________________

*--- Finaliza Registro de ejecución en Bitacora fin_his correcta  ---*
_____________________________________________________________________

Finaliza ejecucion Sun Jul 7 01:27:41 CDT 2019
```

Se puede ver que al principio se ejecuta una carga en la tabla de bitácora, antes de comenzar el proceso y al terminar, nuevamente se hace otra carga en la tabla de bitácora, tal como se describe anteriormente en el código del shell. A continuación, pongo el log de la carga en la tabla de bitácora.

```
BTEQ 14.10.00.11 Sun Jul  7 01:27:39 2019 PID: 30319
+---------+---------+---------+---------+---------+---------+---------+----
.set ERROROUT STDOUT;
 *** Error messages now directed to STDOUT.
+---------+---------+---------+---------+---------+---------+---------+----
.logon
 *** Logon successfully completed.
 *** Teradata Database Release is 13.10.07.12
```

```
 *** Teradata Database Version is 13.10.07.12
 *** Transaction Semantics are BTET.
 *** Session Character Set Name is 'ASCII'.
 *** Total elapsed time was 1 second.
+---------+---------+---------+---------+---------+---------+---------+----
.SET WIDTH 1000;
+---------+---------+---------+---------+---------+---------+---------+----
.if errorlevel <> 0 then .quit 20
+---------+---------+---------+---------+---------+---------+---------+----
-- Limpia datos antes de realizar carga --
DELETE FROM BD_CONTROL.TABLA_BITACORA_HIST
WHERE FCH_PROCESO = CAST('20190707' AS DATE FORMAT 'YYYYMMDD');

 *** Delete completed. No rows removed.
 *** Total elapsed time was 1 second.
+---------+---------+---------+---------+---------+---------+---------+----
-- Inserta a tabla histórica ---
INSERT INTO CONTROL.TABLA_BITACORA_HIST
SELECT
   NUM_PROCESO
  ,NOM_PROCESO
  ,FCH_PROCESO
  ,FCH_INI_CARGA
  ,FCH_FIN_CARGA
  ,RUTA_SCRIPT_LOG
  ,NOM_SCRIPT_LOG
  ,STT_CARGA
FROM CONTROL.TABLA_BITACORA;

 *** Insert completed. 2350 rows added.
 *** Total elapsed time was 1 second.
+---------+---------+---------+---------+---------+---------+---------+----
if errorcode <> 0 then .quit errorcode;
 *** Warning: Expected a '.' before the command
+---------+---------+---------+---------+---------+---------+---------+----
.logoff
 *** You are now logged off from the DBC.
+---------+---------+---------+---------+---------+---------+---------+----
 *** Warning: EOF on INPUT stream.
 *** BTEQ exiting due to EOF on stdin.

 *** Exiting BTEQ...
 *** RC (return code) = 0
```

Este log muestra primero la conexión que hace el bteq con la línea *.logon* la cual debería de mostrar el host, el usuario y la contraseña de la base de datos, aunque se eliminaron estos datos del log por ser datos sensibles.

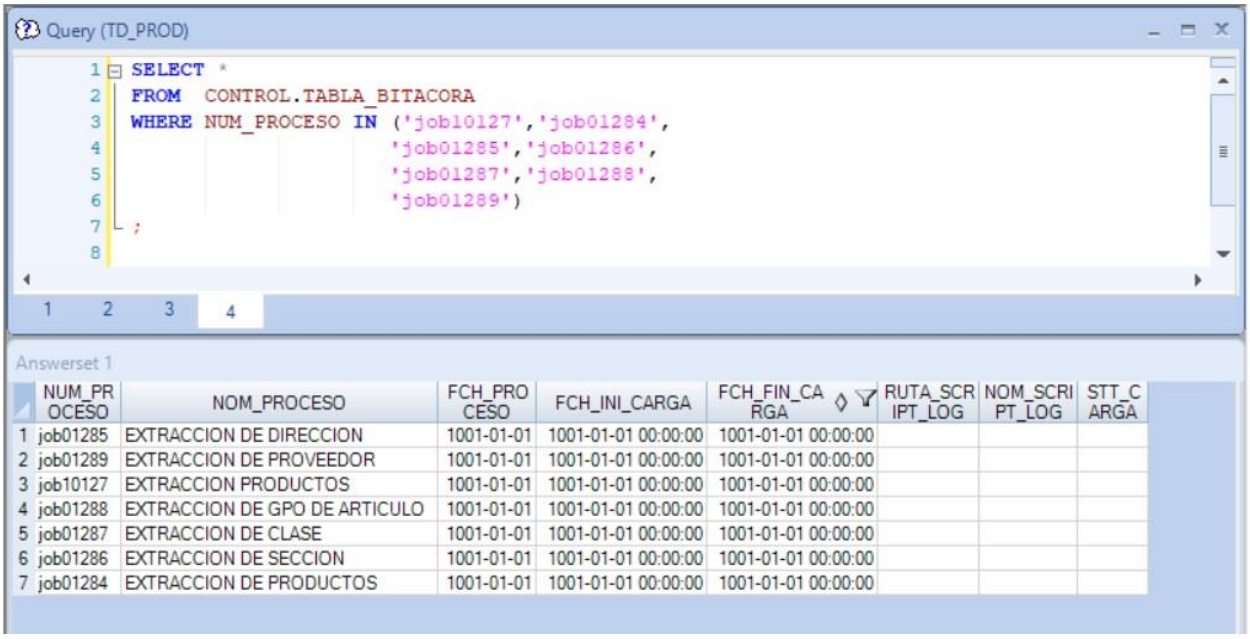
Una vez que se conecta a la base de datos, se hace un *delete* a la tabla esto es por si ya se había ejecutado el proceso y había terminado con errores en la carga, después se hace el insert a la tabla de bitácora histórica. Al final se valida la carga de este proceso obteniendo un código de retorno, el cual sirve para validar el estado del proceso en shell.

Al terminar este proceso comienza la actualización de la tabla de bitácora, siendo ejecutada por el shell job00005.sh el cual es dependiente siempre de la carga a la bitácora histórica, el código del bteq que hace esto es el siguiente.

```
.SET ERROROUT STDOUT;
.LOGON ${var_carga}
.SET WIDTH 1000;
.IF errorlevel <> 0 THEN .quit 20


-- update a bitacora ---
UPDATE ${NOM_DB}.${NOM_TB} SET
      FCH_PROCESO = '1001-01-01',
      FCH_INI_CARGA = '1001-01-01 00:00:00',
      FCH_FIN_CARGA = '1001-01-01 00:00:00',
      RUTA_SCRIPT_LOG = '',
      NOM_SCRIPT_LOG = '',
      STT_CARGA = '',
WHERE NB_PROCESO NOT IN ('job00001', 'job00002', 'job00003',
                         'job00004', 'job00005', 'job00006', 'job10001');
```

Las primeras líneas son independientes de lo que se requiera hacer y es para conectarse a la base de datos, después de la conexión se hace el *update* a la tabla de bitácora sin contar a los procesos de carga inicial, en la figura 12 se muestra cuando la tabla de bitácora ya fue reiniciada.

Los procesos de carga inicial no se reinician porque al hacer las primeras pruebas, nos dimos cuenta de que, al reiniciar toda la tabla estos procesos terminaban sin datos de ejecución y se tuvo que hacer un nuevo sprint para corregir esto.



	NUM_PROCESO	NOM_PROCESO	FCH_PROCESO	FCH_INI_CARGA	FCH_FIN_CARGA	RUTA_SCRIPT_LOG	NOM_SCRIPT_LOG	STT_CARGA
1	job01285	EXTRACCION DE DIRECCION	1001-01-01	1001-01-01 00:00:00	1001-01-01 00:00:00			
2	job01289	EXTRACCION DE PROVEEDOR	1001-01-01	1001-01-01 00:00:00	1001-01-01 00:00:00			
3	job10127	EXTRACCION PRODUCTOS	1001-01-01	1001-01-01 00:00:00	1001-01-01 00:00:00			
4	job01288	EXTRACCION DE GPO DE ARTICULO	1001-01-01	1001-01-01 00:00:00	1001-01-01 00:00:00			
5	job01287	EXTRACCION DE CLASE	1001-01-01	1001-01-01 00:00:00	1001-01-01 00:00:00			
6	job01286	EXTRACCION DE SECCION	1001-01-01	1001-01-01 00:00:00	1001-01-01 00:00:00			
7	job01284	EXTRACCION DE PRODUCTOS	1001-01-01	1001-01-01 00:00:00	1001-01-01 00:00:00			

Figura 12. Tabla de bitácora reseteada (Asistente de Teradata)

Los shells de cada etapa los diseñé para que sólo se tuviera que modificar lo mínimo, por lo que la única diferencia entre éste y el anterior es el número de job a ejecutar, ya que todo se hace por medio de parámetros lo cuales el mismo shell va calculando.

En la tabla 3 están todos los valores que se pueden cargar en la tabla de bitácora, estos valores son para todos los procesos de carga en el DWH.

Descripción en la tabla de bitácora
Error diferencia en registros
Error no existe archivo de carga
Error diferencia en archivo de cifras
Error en la carga
Inicio en la carga
Procesando la carga
Proceso terminado

Tabla 3. Descripción de los estados en la tabla de Bitácora (Diseño propio)

Todos los procesos comienzan con el estado de *Inicio en la Carga*, que es cuando el proceso comienza su ejecución, para inmediatamente pasar al estado de *Procesando la Carga* que es cuando el proceso comienza a realizar alguna actividad, este estado cambia cuando el proceso termina de forma correcta, con lo cual pasa a *Proceso Terminado* o cuando termina en error con alguno de los estados de Error, los estados de error se explicarán en las siguientes etapas de carga del DWH.

Creación de un archivo de control de usuarios

Este proceso ayuda a crear un archivo de control en la ruta de temporales (tmp), el cual obtiene todos los usuarios, contraseñas, la fase de carga del DWH y por último es el sistema fuente o área sujeto al que pertenece. Estas últimas dos columnas son necesarias, ya que son las que ayudan a obtener el usuario correcto en todos los procesos de carga, esto debido a que hay usuarios que pertenecen a la misma etapa del DWH pero que pertenecen a otro sistema fuente. En la tabla 4 están todas las fases de carga en el DWH que existen en el archivo creado de control de usuarios.

Clave de la fase	Descripción de la fase
ini	Proceso de carga inicial
stg	Proceso de carga a staging / de extracción
hom	Proceso de carga a homologación
dwi	Proceso de carga a imagen (DWI)
dwh	Proceso de carga a DWH
sem	Proceso de carga a semántica

Tabla 4. Fases de los procesos de carga del DWH (Diseño propio)

La única fase que no existe como tal, es la de extracción la cual se explicará en su momento más adelante. El shell que ejecuta este proceso es el job00001.sh, lo que este proceso ejecuta es un script de Teradata llamado fast export el cual lee los datos de una tabla y los baja a un archivo físico, el código del fast export que hace esto es el siguiente.

```
.LOGTABLE ${DBWRKPREFIX}.${NOM_TB}_LOG;

.LOGON ${var_carga};

.BEGIN EXPORT SESSIONS 10;

.EXPORT OUTFILE  ${PATH_TMP}.arch_usuario

MODE RECORD FORMAT TEXT;

SELECT CAST(A.texto AS CHAR(45))
FROM (SELECT (TRIM(USR) ||'^'||
              TRIM(PASS)||'^'||
              TRIM(FASE)||'^'||
              TRIM(FTE_AREA_SUJ)) AS texto
FROM ${DBCTLPREFIX}.${NOM_TB}) A;

.END EXPORT;
```

En donde las primeras líneas son parámetros que sirven para ejecutar el código SQL, de las cuales la primera línea es el comando de la tabla de log generado por el proceso, la segunda línea es el comando con el que se conecta a la base de datos, la tercera línea es el número de sesiones que le damos al proceso, la cuarta línea es el archivo que se va a crear, la quinta línea es el formato que va a tener el archivo y las demás líneas son el query utilizado para crear el archivo, al igual que los scripts anteriores, se usan parámetros para no tener en los scripts datos sensibles, esta característica se tiene en todos los scripts y shells de carga.

Creación de un archivo de control de procesos

Este proceso ayuda a crear un archivo de control en la ruta de temporales (tmp), el cual obtiene todos los procesos que el DWH necesita para poder cargar la información, estos procesos van desde las extracciones hasta la carga a la semántica.

Los procesos de carga inicial no se incluyen, ya que estos procesos ayudan a preparar todo lo necesario para que se pueda ejecutar el DWH. El shell que ejecuta este proceso es el job00002.sh, este proceso se ejecuta también por un fast export y al igual que le proceso anterior, lee los datos de una tabla y los baja a un archivo físico, el código del fast export que hace esto es el siguiente.

```
.LOGTABLE ${DBWRKPREFIX}.${NOM_TB}_log;

.LOGON ${var_carga};

.BEGIN EXPORT SESSIONS 10;

.EXPORT OUTFILE  ${PATH_TMP}.arch_proceso

MODE RECORD FORMAT TEXT;

SELECT CAST(A.texto AS CHAR(1000))
FROM (SELECT (NOM_JOB||'^'||
              NUM_SECUENCIA||'^'||
              NOM_FASE||'^'||
              NOM_FUENTE||'^'||
              NOM_AREA_SUJ||'^'||
              NOM_SCRIPT||'^'||
              NOM_ARCHIVO_DATOS||'^'||
              NOM_ARCHIVO_CIFRAS||'^'||
              NOM_BD_DESTINO||'^'||
              NOM_OBJ_DESTINO||'^'||
              NOM_BD_FUENTE||'^'||
              NOM_OBJ_FUENTE) AS texto
FROM  ${DBCTLPREFIX}.${NOM_TB}) A
ORDER BY  A.NOM_JOB, A.NUM_SECUENCIA
;

.END EXPORT;
```

Al igual que el script anterior, las primeras líneas cinco líneas son los parámetros que sirven para ejecutar el código SQL y las demás líneas, son el query utilizado para crear el archivo. Este archivo al igual que el anterior, son los más importantes porque con ambos se ejecutan todos los shells para la carga en el DWH en todas sus etapas. Es por esto, que al hacer nuevos desarrollos se crean scripts de carga a las tablas de usuarios y de procesos.

El archivo de los usuarios no es difícil de generar el script de carga a su respectiva tabla, ya que sólo tiene cuatro columnas y cada una es muy diferente de la otra, pero en el archivo de procesos hay columnas muy parecidas, por lo que es necesario el tener un mejor control de los parámetros los cuales a continuación describo.

- NOM_JOB. Es el nombre del proceso que se ejecutará

- NUM_SECUENCIA. Es la secuencia del shell que se ejecutará
- NOM_FASE. Es el nombre de la fase de carga del DWH que se ejecutará
- NOM_FUENTE. Es el sistema fuente del que proviene la interfaz
- NOM_AREA_SUJ. Es el área sujeto al que pertenece el DWH
- NOM_SCRIPT. Nombre del script usado para realizar alguna carga
- NOM_ARCHIVO_CARGA. Nombre del archivo que se debe de carga en el DWH
- NOM_ARCHIVO_CIFRAS. Nombre del archivo de cifras de control
- NOM_BD_DESTINO. Nombre de la BD destino del proceso
- NOM_OBJ_DESTINO. Nombre del objeto de la BD destino del proceso
- NOM_BD_FUENTE. Nombre de la BD origen del proceso
- NOM_OBJ_FUENTE. Nombre del objeto de la BD origen del proceso

En etapas posteriores de carga en el DWH, las columnas del archivo de datos serán explicadas, ya que no todas las columnas son ocupadas en cada etapa del proceso de carga.

Depuración de interfaces y logs de carga

Este proceso ayuda a depurar las interfaces y los logs de carga, dejando siempre los últimos cuatro logs, así como las interfaces de carga, también en el proceso de las interfaces además de depurar las interfaces mueve a la ruta de procesados. Este proceso es muy importante porque mantiene el espacio del servidor Linux, que es donde se depositan todas las interfaces cargadas en el DWH.

Este proceso es ejecutado por el shell job00003.sh, el depurado se hace por medio de scripts a nivel de sistema operativo y se encuentran dentro del shell. El código que realiza estas acciones es el siguiente.

```
# Variables para el depurado

PATH_RAIZ=/datastage_data/empresa_A/DWH
PATH_INPUT=${PATH_RAIZ}/input/
PATH_LOGS=${PATH_RAIZ}/logs/
PATH_FUENTE=${PATH_RAIZ}/procesados/
```

```
FECHA_BORR=`date +%Y%m%d --date='-5 day'`
FECHA_BORR_LOG=`date +%Y%m%d --date='-5 day'`
FECHA_BORR_PROC=`date +%Y%m%d --date='-2 day'`
FECHA=`date "+%Y%m%d"`

# Mueve archivos de input a procesados

echo "Fecha de depuración para input a procesados: $FECHA_BORR_PROC"
for j in $(ls ${PATH_INPUT} ); do
      if [ -d ${PATH_INPUT}${j} ]; then
            echo "     Depurando Directorio: $j     "
            PATH_ORIG=$PATH_INPUT${j}
            echo "Archivos movidos: "
            for i in $(ls ${PATH_ORIG} ); do
                  FECHA_ACT=`echo  $i | cut -d"." -f2`
                  if [ ${FECHA_ACT} -lt ${FECHA_BORR_PROC} ]; then
                  echo $i
                  mv ${PATH_ORIG}/${i}  $PATH_FUENTE${j}
                  gzip $PATH_FUENTE${j}/${i}
                  fi
            done
      fi
done

# Elimina archivos de procesados

echo "Fecha de depuración para input: $FECHA_BORR"
for j in $(ls ${PATH_FUENTE} ); do
      if [ -d ${PATH_FUENTE}${j} ]; then
            echo "     Depurando Directorio: $j     "
            PATH_ORIG=$PATH_FUENTE${j}
            k=0
            echo "Archivos eliminados: "
            for i in $(ls ${PATH_ORIG} ); do
                  FECHA_ACT=`echo  $i | cut -d"." -f2`
                  if [ ${FECHA_ACT} -lt ${FECHA_BORR} ]; then
                        let k=$k+1
                        echo $i
                        #Comienza el borrado de archivos
                        rm ${PATH_ORIG}/${i}
                  fi
            done
            let k=$k+$k
      fi
done

# Depura logs antiguos

echo "     Depurando logs     "
l=0
for i in $(ls ${PATH_LOGS} ); do
      if [  -f ${PATH_LOGS}${i} ]; then
      FECHA_ACT_LOG=`echo  $i | cut -d"." -f2`
            if [ ${FECHA_ACT_LOG} -lt ${FECHA_BORR_LOG} ]; then
                  let l=$l+1
```

```
                echo "Log eliminado: "
                echo $i
                rm ${PATH_LOGS}/${i}
            fi
        fi
done
```

Para explicar este proceso, se manejan tres directorios diferentes los cuales son, en donde las interfaces son depositadas por los sistemas fuentes que alimentan al DWH, en el directorio de input, en donde están las interfaces que ya fueron procesadas, en el directorio de procesados y finalmente en donde los logs generados por todos los procesos de carga, en el directorio de logs.

En el directorio de input se encuentran los subdirectorios que pertenecen a cada uno de los sistemas fuente, en estos subdirectorios siempre están las interfaces que se cargarán al DWH y las interfaces que se cargaron el día anterior.

La primer parte del código lo que hace es mover las interfaces de input a procesados, teniendo en cuenta las interfaces que tengan en el nombre la fecha del día de proceso a menos 2 días, es decir si el proceso se ejecuta el 31 de Julio de 2019, los archivos que debe mover son los que tenga en su nombre la fecha de 20190728, finalmente cada que mueve un archivo, éste es comprimido para mantener el espacio en el file system.

La segunda parte del código lo que hace es eliminar las interfaces que tenga en el nombre la fecha del día de proceso a menos 5 días en el directorio de procesados, para mantener siempre las últimas 5 interfaces que han sido procesadas.

Por último, lo que el código hace es eliminar los archivos de logs que tengan en el nombre la fecha del día de proceso a menos 5 días en el directorio de logs, originalmente se quería guardar los últimos 30 archivos, pero por temas de espacio en el servidor se redujo la cantidad a sólo 5 archivos.

Reinicio de los jobs de DataStage

En la ejecución de los procesos de carga de DWH, se usan jobs de DataStage los cuales se dividen en jobs de secuencias, subsecuencias, extracción y transformación, todos los jobs son ejecutados por medio de shells.

Este proceso ayuda a reiniciar los jobs de DataStage, esto es debido a que puede haber varias razones por las que los procesos de carga no finalicen, pero hay ocasiones en que el problema de la carga se debe a la información proporcionada por el sistema fuente, por lo que cuando se reporta este error hay ocasiones en que el problema no se solucionará el mismo día.

Esto también ayuda en el monitoreo de los procesos batch, ya que una de las mejoras que implementé, fue que las secuencias de DataStage cuando fallaran se pudieran ejecutar de nueva cuenta, esto para no tener la necesidad de tener interacción humana si se deben de ejecutar de nuevo, después de un fallo en su proceso.

Esto se hizo, debido a que las secuencias que ya existían para otros procesos no tenían esta funcionalidad, por lo que siempre era necesario el tener que reiniciar la secuencia, pero esto tenía dos inconvenientes.

- El tener que reiniciar la secuencia de manera manual cada vez que se requiera.
- Si el proceso en cuestión estaba por terminar, como se había reiniciado la secuencia debía de volver a ejecutar todo.

Ahora ya no hay necesidad de hacer esta acción, sólo se necesita ejecutar de nueva cuenta el proceso que tuvo una falla y reinicia desde donde falló, como se observa en la figura 13.

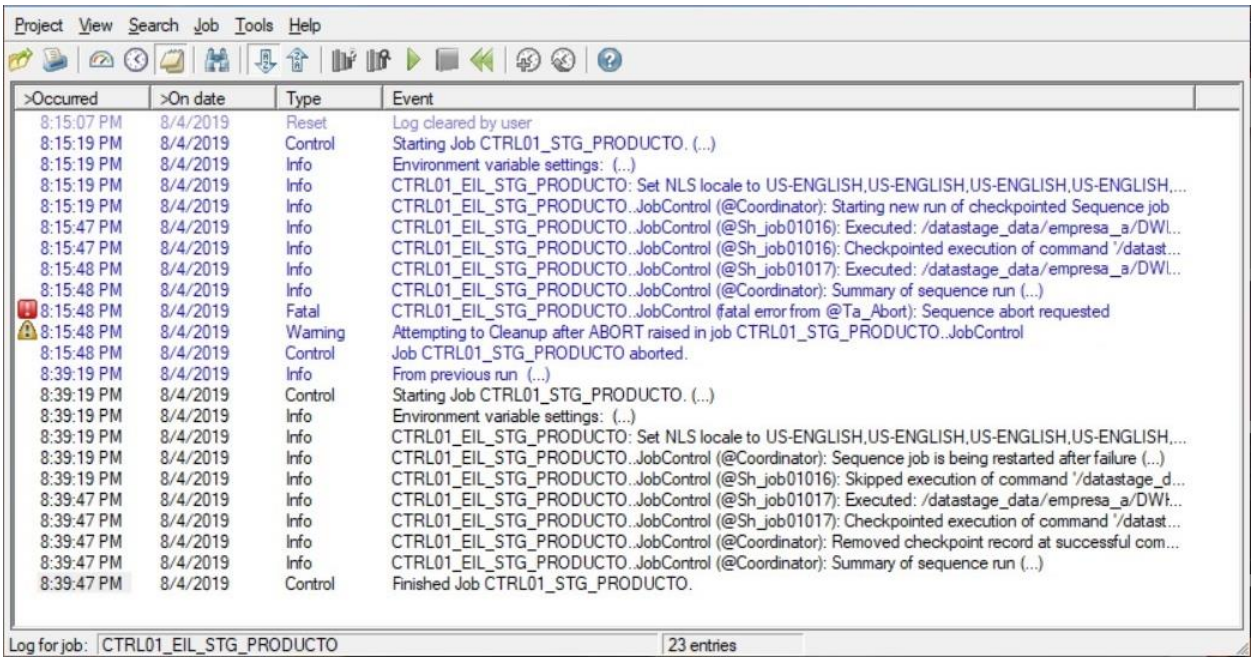


Figura 13. Re-ejecución de un proceso de carga del DWH (Log de ejecución DataStage)

Como se puede ver, en la primera ejecución corrió el primer shell job01016.sh y falló el segundo el shell job01017.sh, una vez que se arregla lo que falló del segundo proceso se ejecuta de nuevo y ya sólo es ejecutado el segundo shell.

Esta es la razón por la que este proceso existe, tomando de ejemplo la figura anterior si el proceso no hubiera podido ser arreglado, en la ejecución del día siguiente sólo hubiera sido ejecutado el segundo proceso y se hubiera saltado el primero. Este proceso es ejecutado por el shell job00006.sh, lo cual se hace por medio de comandos a nivel de sistema operativo, el código que ejecuta este reinicio de los jobs es el siguiente.

```
echo "     Realizando resets a los jobs de DataStage     "

# Inicia el proceso de reset
# Productos
cd /opt/IBM/InformationServer/Server/DSEngine/
. ./dsenv
/opt/IBM/InformationServer/Server/DSEngine/bin/dsjob -run -mode RESET
empresa_A CTRL01_EXT_PRODUCTO
echo 'Se reinicio el Job CTRL01_EXT_PRODUCTO'
```

```
cd /opt/IBM/InformationServer/Server/DSEngine/
. ./dsenv
/opt/IBM/InformationServer/Server/DSEngine/bin/dsjob -run -mode RESET
empresa_A CTRL01_STG_PRODUCTO
echo 'Se reinicio el Job CTRL01_STG_PRODUCTO'

cd /opt/IBM/InformationServer/Server/DSEngine/
. ./dsenv
/opt/IBM/InformationServer/Server/DSEngine/bin/dsjob -run -mode RESET
empresa_A CTRL00_DWH_PRODUCTO
echo 'Se reinicio el Job CTRL00_DWH_PRODUCTO'

cd /opt/IBM/InformationServer/Server/DSEngine/
. ./dsenv
/opt/IBM/InformationServer/Server/DSEngine/bin/dsjob -run -mode RESET
empresa_A CTRL01_DWH_PRODUCTO
echo 'Se reinicio el Job CTRL01_DWH_PRODUCTO'

cd /opt/IBM/InformationServer/Server/DSEngine/
. ./dsenv
/opt/IBM/InformationServer/Server/DSEngine/bin/dsjob -run -mode RESET
empresa_A CTRL02_DWH_PRODUCTO
echo ' Se reinicio el Job CTRL02_DWH_PRODUCTO'
```

Este proceso usa sólo tres instrucciones para realizar el reset de cada uno de los jobs de DataStage, con la primera instrucción se cambia al directorio de DataStage para realizar el reset, la segunda instrucción es para dar de alta las variables para realizar el proceso de reset y finalmente, con la tercera instrucción se hace el reset del job de DataStage.

El único cambio que se hace en esa instrucción es después de la palabra RESET, ya que ahí va el nombre del proyecto de DataStage en donde se localizan los jobs, debido a que su nombre es el de la empresa de Retail, se cambió por empresa_A.

Estos procesos se ejecutan mediante un shell general con la numeración de job1000x.sh, estos procesos son los que ejecutan desde Control – M, esto lo diseñé de esta manera para simplificar la cantidad de procesos que se ejecutarán en la malla del DWH, de otra forma si cada proceso que realiza alguna acción se metiera en la malla quedaría prácticamente imposible de darle mantenimiento.

Al igual que los shells anteriores este shell se carga mediante parámetros, por lo que sólo se hacen algunos cambios entre cada shell, el código que los procesos internos es el siguiente.

```
NUM_JOB=job10001
NOM_DATASTAGE=SECUENCIA_INICIAL_DWH
# Arreglo Procesos
log[0]=${PATH_LOGS}job00001.${FECHA}.log
log[1]=${PATH_LOGS}job00002.${FECHA}.log
log[2]=${PATH_LOGS}job00003.${FECHA}.log
log[3]=${PATH_LOGS}job00004.${FECHA}.log
log[4]=${PATH_LOGS}job00005.${FECHA}.log
log[5]=${PATH_LOGS}job00006.${FECHA}.log
# Subrutina para ejecucion de DataStage
EjecutaDatastage()
{
      echo "    Comienza proceso: `date`.      " > $ARCH_LOGS
      echo "_____________________________________" > $ARCH_LOGS
      echo "    Proceso de Flujo Inicio        " >> $ARCH_LOGS
      echo "Ejecuta la secuencia de DataStage "${NOM_DATASTAGE} >> $ARCH_LOGS
      # Ejecuta insercion de inicio a la bitacora
      cd /opt/IBM/InformationServer/Server/DSEngine/
      . ./dsenv
      /opt/IBM/InformationServer/Server/DSEngine/bin/dsjob -run ${ov_PARAMS} -
jobstatus empresa_A ${NOM_DATASTAGE}
      status=$?
      # Evaluacion de la corrida
      if [ $status  = 1 ] || [ $status = 2 ]; then
            for i in ${log[@]}
                  do
                  if [  -f ${i} ]; then
                        PROCESO=`echo "$i" |cut -d"/" -f6 | cut -c1-8`
                        echo "_____________________________" >> $ARCH_LOGS
                        echo " EJECUCION  C O R R E C T A" >> $ARCH_LOGS
                        echo "      JOB:  $PROCESO        " >> $ARCH_LOGS
                  fi
            done
            echo "Termina el proceso "${NOM_DATASTAGE}" EXITOSAMENTE" >>
$ARCH_LOGS
            statusFinal=0
      else
            for i in ${log[@]}
                  do
                  if [  -f ${i} ]; then
                        PROCESO=`echo "$i" |cut -d"/" -f6 | cut -c1-8`
                        open=`grep "Se cancela" $i  |tr '[' ' '|tr ']' ' '`
                        if [ "${open}" != "" ]; then
                              echo "_____________________________" >> $ARCH_LOGS
                              echo "        E R R O R            " >> $ARCH_LOGS
                              echo "      JOB:  $PROCESO         " >> $ARCH_LOGS
                              cat $i
                        else
                              echo "_____________________________" >> $ARCH_LOGS
                              echo "EJECUCION  C O R R E C T A " >> $ARCH_LOGS
```

```
                              echo "      JOB:  $NUM_JOB       " >> $ARCH_LOGS
                        fi
                  fi
                  done
                  if [ $status = 3 ]; then
                        echo "Termina el proceso "${NOM_DATASTAGE}" con ABORT"
>> $ARCH_LOGS
                        statusFinal=1
                  else
                        echo "Termina el proceso "${NOM_DATASTAGE}" con
ERRORES" >> $ARCH_LOGS
                        statusFinal=1
            fi
            # Agrega el log de los procesos del arreglo de procesos
                  if [  -f ${log[0]} ]; then
                        echo " " >> $ARCH_LOGS
                        cat ${log[0]} >> $ARCH_LOGS
                  fi
                  if [  -f ${log[1]} ]; then
                        echo " " >> $ARCH_LOGS
                        cat ${log[1]} >> $ARCH_LOGS
                  fi
                  if [  -f ${log[2]} ]; then
                        echo " " >> $ARCH_LOGS
                        cat ${log[2]} >> $ARCH_LOGS
                  fi
                  if [  -f ${log[3]} ]; then
                        echo " " >> $ARCH_LOGS
                        cat ${log[3]} >> $ARCH_LOGS
                  fi
                  if [  -f ${log[4]} ]; then
                        echo " " >> $ARCH_LOGS
                        cat ${log[4]} >> $ARCH_LOGS
                  fi
                  if [  -f ${log[5]} ]; then
                        echo " " >> $ARCH_LOGS
                        cat ${log[5]} >> $ARCH_LOGS
                  fi
      fi
      echo "Termina proceso: `date`." >> $ARCH_LOGS
      exit ${statusFinal}
}
```

En este proceso las primeras dos líneas de código cambian en cada shell job1000x.sh que se crea, la primera línea es el shell que se va a ejecutar, además sirve también para la tabla de bitácora cual proceso es el que se ejecuta, esta línea existe en todos los shells. La segunda línea es para saber el proceso de DataStage que debe ser ejecutado, esta instrucción sólo existe en estos shells y los de extracción los cuales se verán más adelante.

El arreglo de procesos, son todos los procesos que internamente son ejecutados en el job de DataStage, esto varia ya que cada job de DataStage ejecuta un número de shells diferente, en este caso son los 5 procesos que anteriormente se analizaron. En la función creada para la ejecución de DataStage, después de poner el comentario de inicio de proceso, se evalúa la ejecución del job de DataStage, teniendo los siguientes estatus posibles

- Estatus = 1. Cuando la ejecución de un job de DataStage termina correctamente
- Estatus = 2. Cuando la ejecución de un job de DataStage termina con warnings
- Estatus = 3. Cuando la ejecución de un job de DataStage termina con abort
- Estatus > 3. Cuando la ejecución de un job de DataStage termina con errores

Cuando el estatus es 1 ó 2, indica que la secuencia terminó bien y agregamos los procesos ejecutados que terminaron de forma correcta gracias al ciclo for, si por el contrario el estatus es 3, indica que la secuencia terminó con abort y si terminó con un estatus mayor a 3, indica que la secuencia terminó con errores. La diferencia entre el estatus 3 y uno mayor a 3 es lo siguiente, para el estatus 3 los errores son porque algún job fallo en su ejecución, en cambio para el estatus mayor a 3 es porque algún job tiene un estado con el que no puede ejecutarse o también puede ser que no exista dicho job.

Cuando sucede que el job termina con abort o con errores, se agregan los logs que fueron ejecutados internamente, en este caso los procesos vistos anteriormente, por esta razón se hace el arreglo de procesos y por lo mismo, cada shell de este tipo tiene un arreglo diferente. A continuación, se agrega un log con una ejecución correcta.

```
Comienza proceso: Sun Jun 30 06:04:05 CDT 2019.
_____________________________________
    Proceso de Flujo Inicio
Ejecuta la secuencia de DataStage SECUENCIA_INCIAL_DWH
_____________________________
EJECUCION  C O R R E C T A
      JOB:  job00001
_____________________________
EJECUCION  C O R R E C T A
      JOB:  job00002
_____________________________
```

```
EJECUCION  C O R R E C T A
      JOB:  job00003

_____________________________
EJECUCION  C O R R E C T A
      JOB:  job00004

_____________________________
EJECUCION  C O R R E C T A
      JOB:  job00005

Termina el proceso SECUENCIA_INICIAL_DWH
Termina proceso: Sun Jun 30 06:13:31 CDT 2019.
```

Esta etapa además de ser la que hace los arreglos necesarios para que todos los procesos puedan ser ejecutados, también es la que da condición a todos los procesos de extracción de interfaces de los sistemas fuentes o de carga a staging, siendo estos dos procesos los que comienzan con la carga de la información en el DWH.

Extracción de interfaces de los sistemas fuente

Esta etapa originalmente no se tenía contemplada, ya que cuando tuve las primeras reuniones con las diversas áreas que son las dueñas de la información, se acordó que cada sistema fuente que alimentaría al DWH nos enviarían las interfaces requeridas.

Sin embargo, esto no se logró con todos los sistemas fuente, ya que varios sistemas fuentes no tenían el personal para poder atender los requerimientos solicitados para el envío interfaces. Por lo que tuvo que implementarse esta nueva etapa en el proceso de carga al DWH.

Esta fase la diseñé para que pudiera ser una nueva etapa del proceso, teniendo en cuenta los mismos estándares que fueron solicitados a los sistemas fuentes, en cuanto al envío de las interfaces, por lo que se crearon nuevos subdirectorios en el directorio de input con el nombre de los nuevos procesos de extracción.

Al igual que en los procesos enviados por los sistemas fuente, el nombre de estos nuevos subdirectorios es el mismo nombre que el sistema fuente del cual pertenecen. En esta etapa al igual que la anterior se usan shells para ejecutar estos procesos, los cuales están representados en la figura 14.

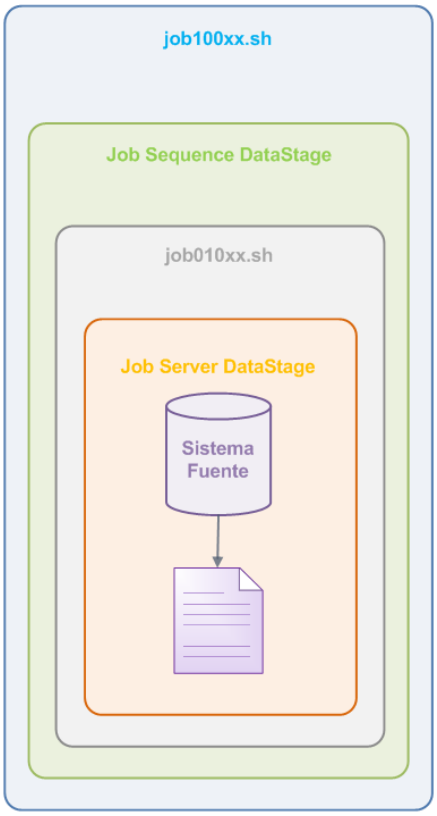


Figura 14. Proceso de Extracción (Diseño propio)

Al igual que en el proceso anterior, los shells que ejecutan los procesos de extracción son los job1000x.sh, los cuales ya fueron explicados. Estos shells lo que ejecutan son secuencias de DataStage conocidos como job Sequence, las cuales contienen los shells job0100x.sh que ejecutan cada proceso de extracción, estas extracciones se hacen mediante DataStage con jobs de tipo Server.

A continuación, se explica cómo funcionan cada tipo de job de DataStage y la gran utilidad que se tiene en cada proceso, para este proceso el job Server que se ocupa es para realizar las extracciones de cada una de las interfaces que se necesitan. En la figura 15 se observa uno de estos Jobs.

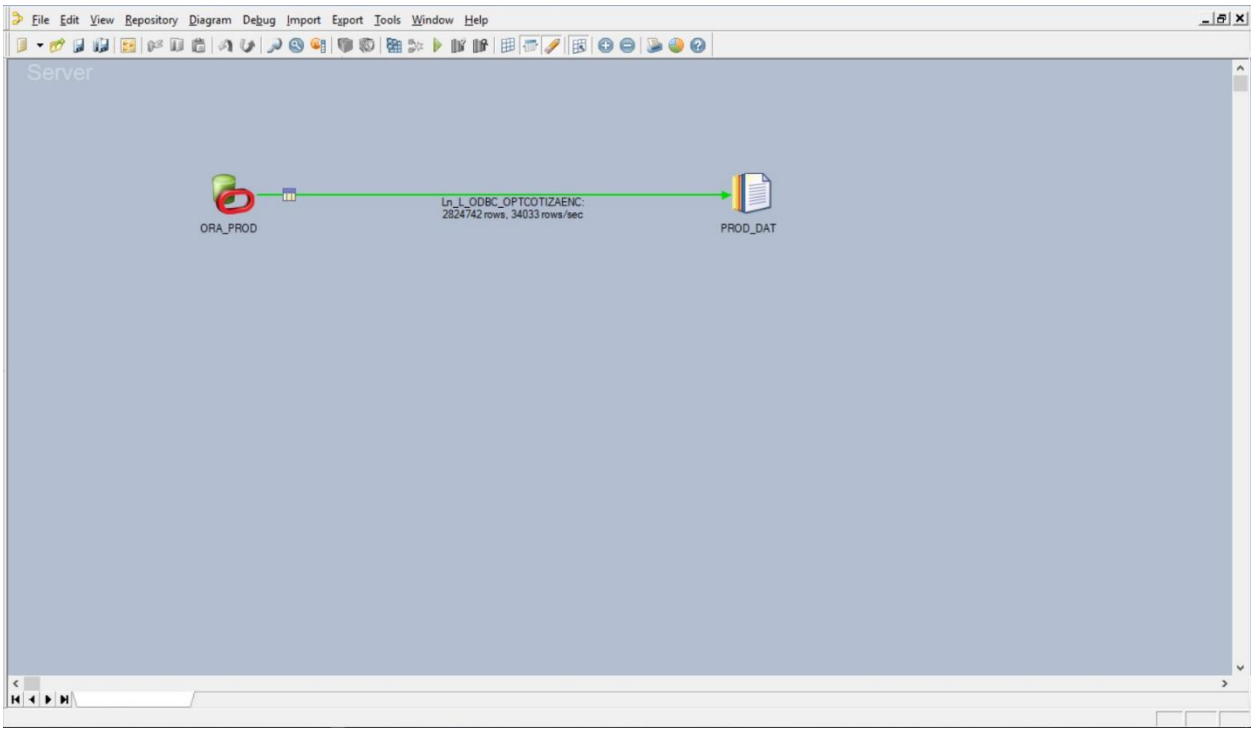


Figura 15. Job Server de Extracción (Designer de DataStage)

Este tipo de job contiene dentro de sus características un módulo de conectores de distintas bases de datos, por lo que es muy útil debido a que la gran mayoría de los sistemas fuentes ocupan Oracle como su base de datos. Por lo que, sólo es necesario pasarle al conector los parámetros necesarios de la base de datos a la cual se requiere acceder.

Además, dependiendo de las columnas que se desean extraer, el query de extracción puede ser por cada una de las columnas deseadas o por todas las columnas de la tabla, teniendo las mismas características como si se usara un manejador de base de datos. También se tiene un módulo de archivos, del que se ocupa un Stage[6] llamado Sequential File el cual es para ocupar archivos de tipo texto, teniendo como ventaja que se puede dar el formato deseado de como se requiere el archivo, como puede ser el tipo de separador de columna.

[6] Es como se les conoce a los componentes de DataStage. Ver anexo

En este Stage se puede indicar el formato, la ruta y el nombre del archivo, teniendo con esto, los mismos estándares que se tienen para las interfaces que los sistemas fuente envían para la carga del DWH.

Este tipo de proceso se crea con la ayuda de cada sistema fuente, en este caso será el módulo de productos, conocido en la empresa de Retail como jerarquía de productos, teniendo los siguientes queries de extracción

- Catálogo de clase
- Catálogo de dirección
- Catálogo de grupo de artículo
- Catálogo de sección
- Dimensión de productos
- Dimensión de proveedor

A continuación, se revisa el código de cada uno de estos procesos, los cuales el sistema fuente dio las reglas de cómo son las extracciones. El query del catálogo de clase es el siguiente

```
SELECT
      CCLASE    AS CLASE_PRODUCTO_CVE,
      DCLASE    AS CLASE_PRODUCTO_DES,
      CDEPTO    AS GPO_ART_PRODUCTO_CVE
FROM  TCLASE
WHERE FECHBAJA = '00000000'
;
```

Al revisar el query, se observa que el campo de fecha es de tipo caracter, por lo que ya se observa que los datos no serán lo mismo. El query del catálogo de dirección es el siguiente

```
SELECT
      CDIVISION AS DIRECCION_PRODUCTO_CVE,
      DDIVISION AS DIRECCION_PRODUCTO_DES
FROM  TDIVISION
WHERE FECHBAJA = '00000000'
;
```

Al revisar el query, también se observa que el campo de fecha es de tipo caracter, viendo que esto comienza a ser una constante. El query del catálogo de grupo de artículo es el siguiente

```
SELECT
      CDEPTO      AS GPO_ART_PRODUCTO_CVE,
      DDEPTO      AS GPO_ART_ PRODUCTO_DES,
      CSECCION    AS SECCION_PRODUCTO_CVE,
      CDIVISION   AS DIRECCION_PRODUCTO_CVE
FROM  TGPOARTICULO
WHERE FECHBAJA = '00000000'
;
```

Al revisar el query, hay campos que se ocupan para poder homologar la estructura actual con la estructura de esta nueva empresa. El query del catálogo de sección es el siguiente

```
SELECT
      CSECCION     AS SECCION_PRODUCTO_CVE,
      DSECCION     AS SECCION_PRODUCTO_DES,
      CDIVISION    AS DIRECCION_PRODUCTO_CVE
FROM  TSECCION
WHERE FECHBAJA = '00000000'
;
```

Al revisar cada uno de los queries anteriores, se observa que las columnas de fecha son de tipo carácter, pero debido a que estas columnas sólo son una condición para obtener los datos, no tiene mayor implicación en los procesos. El query de la tabla de productos es el siguiente

```
SELECT
      B.CSKGEAN     AS PRODUCTO_CVE,
      A.DSKCEAN     AS PRODUCTO_DES,
      SUBSTR(A.FALTA,1,4)||'-'||SUBSTR(A.FALTA,5,2)||'-'||SUBSTR(A.FALTA,7,2)
AS FECHA_PRODUCTO_ALTA,
      A.CDEPTO      AS GPO_ART_PRODUCTO_CVE,
      A.CPROVEDOR   AS PROVEEDOR_CVE,
      A.ICOSTO      AS PRECIO_COSTO,
      A.IPVTA       AS PRECIO_VENTA,
      A.TALLA       AS TALLA,
      A.CCOLOR      AS COLOR_CVE,
      A.DCOLOR      AS COLOR_DES,
      A.DSMARCA     AS MARCA,
      A.CCLASE      AS CLASE_CVE
```

```
FROM  TPRODCUTO A,
      TVALIDA_PRODUCTO B
WHERE A.CVEVAL = B.CVEVAL
AND    (A.FALTA = '#FECHA#' OR A.FMOD = '#FECHA#')
;
```

Al revisar el query se tienen las siguientes observaciones, debido a que en este query sí tenemos en los datos algunas fechas, es necesario poner las fechas en un formato valido.

Esto es debido a que las fechas están con un formato alfanumérico, en lugar de ser el tipo date, por lo que cuando hay fechas que no existen o como en este caso tienen el valor de 000000, se pone una fecha dummy con el valor de 1900-01-01. El query de la tabla de proveedores es el siguiente

```
SELECT
      CVEPROV AS PROVEEDOR_CVE,
      DESPROV AS PROVEEDOR_NOMBRE,
      CASE WHEN SUBSTR(FALTA,1,4)||'-'||SUBSTR(FALTA,5,2)||'-
'||SUBSTR(FALTA,7,2) = '0000-00-00' THEN '1900-01-01'
            ELSE SUBSTR(FALTA,1,4)||'-'||SUBSTR(FALTA,5,2)||'-
'||SUBSTR(FALTA,7,2) END AS PROVEEDOR_FECHA_ALTA,
      CFISCAL AS PROVEEDOR_RFC
FROM  TPROVEEDOR
WHERE (FALTA = '#FECHA#' OR FMOD = '#FECHA#')
;
```

Al revisar este query, se tiene el mismo tratamiento con los campos de fechas, además para ambos queries se tiene un filtro por la fecha de alta y la de modificación, estos valores están representados por una variable #FECHA#, la cual es la única diferencia entre un manejador de base de datos y el DataStage.

Esto se debe a que las variables se representan entre los símbolos de #, estas variables pueden usarse en varios módulos de un job de DataStage. Para el caso de los dos últimos queries, el valor de la variable es transmitida por medio de los que ejecutan los Jobs de extracción.

Estos queries son ejecutados en jobs de extracción, cada uno de estos jobs son ejecutados por medio de shells, por lo que a continuación se analizará parte del código de estos shells

```
NUM_JOB="job01284"
NOM_DATASTAGE="EXT01_PRODUCTO"
INF_FASE=stg
INF_FUENTE=emp_b
NOM_TABLA="EXTRACCION DE PRODUCTOS"
# Obtiene fecha de proceso
ObtenFecha()
{
      if [ -z $1 ]; then
            FECHA="`date +%Y%m%d --date='-1 day'`"
            echo "La fecha es ${FECHA}"
      else
            FECHA=$1
            echo "La fecha es ${FECHA}"
      fi
}
# Ejecuta Job de DataStage
EjecutaDatastage()
{
      echo "Comienza proceso: `date`." > $ARCH_LOGS
      echo " ################################################ " >> $ARCH_LOGS
      echo " #    Proceso de extraccion de ${NOM_TABLA}     # " >> $ARCH_LOGS
      echo " ################################################ " >> $ARCH_LOGS
      echo "Ejecuta el job de DataStage "${NOM_DATASTAGE} >> $ARCH_LOGS
      echo "Ejecuta el numero de proceso: "${NUM_JOB} >> $ARCH_LOGS
      # Parametros para DataStage
      ov_PARAMS="-param FECHA=${FECHA}"
      # Ejecuta secuencia de DataStage
      /opt/IBM/InformationServer/Server/DSEngine/bin/dsjob -run ${ov_PARAMS} -
jobstatus empresa_A ${NOM_DATASTAGE}
      status=$?
      # Evaluacion de la corrida
      if [ $status  = 1 ] || [ $status = 2 ]; then
            echo "Termina el proceso "${NOM_DATASTAGE}" EXITOSAMENTE" >>
$ARCH_LOGS
            statusFinal=0
      else
            if [ $status = 3 ]; then
                  echo "Termina el proceso "${NOM_DATASTAGE}" con ABORT" >>
$ARCH_LOGS
                  statusFinal=1
            else
                  echo "Termina el proceso "${NOM_DATASTAGE}" con ERRORES" >>
$ARCH_LOGS
                  statusFinal=1
            fi
            # Manda el log de DataStage al log del shell
            /opt/IBM/InformationServer/Server/DSEngine/bin/dsjob -logsum
empresa_A ${NOM_DATASTAGE} >> $ARCH_LOGS
```

```
            # Reinicia el job para una futura ejecución
            echo "Reset al job "${NOM_DATASTAGE} >> $ARCH_LOGS
            cd /opt/IBM/InformationServer/Server/DSEngine/
            . ./dsenv
            /opt/IBM/InformationServer/Server/DSEngine/bin/dsjob -run -mode
RESET empresa_A ${NOM_DATASTAGE}
      fi
      echo "El estatus final es "${statusFinal}
      echo "Termina proceso: `date`." >> $ARCH_LOGS
      exit ${statusFinal}
}
```

En este shell, las primeras líneas son las que tienen la información del proceso que debe ejecutarse, como el número de proceso, el nombre del job de DataStage, la fase de carga, el sistema fuente y el nombre del proceso que se insertará en la bitácora.

La función de fecha es la que obtiene el parámetro que se le envía al job de DataStage, siendo el filtro que se usa en los queries de extracción, así como parte del nombre de las interfaces generadas.

La siguiente función es con la que el job de DataStage es ejecutado, al igual que en los shells job1000x.sh analizados anteriormente, se analiza el estatus de ejecución, en donde si el estatus es finalizado sin errores se continúa con los siguientes procesos, en caso contrario el job manda un estado de falla y se reinicia el job para una futura ejecución.

Esta última acción es una de las mejoras que se hicieron en este nuevo proceso de carga al DWH, ya que los procesos que la empresa de Retail tenía en la herramienta de DataStage, siempre tenían que reiniciarse de forma manual como ya se ha explicado anteriormente.

Estos shells de extracción se ejecutan por medio de un job de DataStage de tipo Sequence, el cual sirve para realizar una malla interna de los procesos que deben de ejecutarse como se observa en la figura 16.

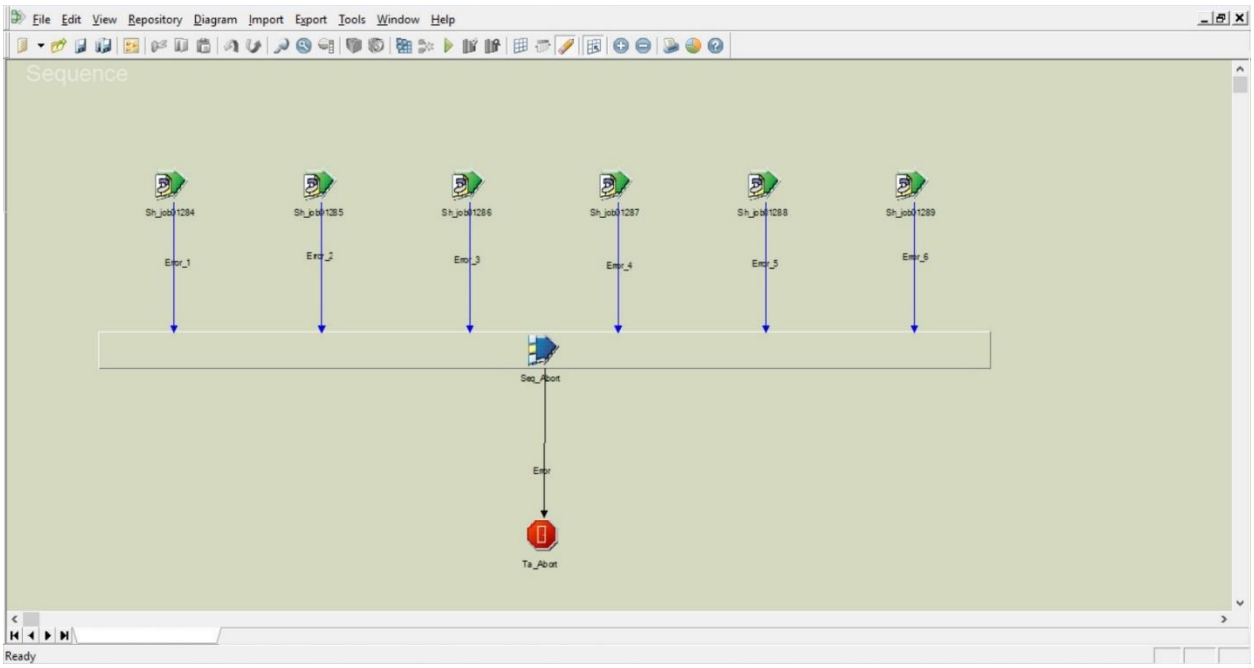


Figura 16. Job Sequence de Extracción (Designer DataStage)

Este tipo de job contiene dentro de sus características un Stage que sirve para ejecutar comandos, el cual es muy útil porque en cada uno de estos Stages, se manda a ejecutar cada uno de los shells de extracción, en donde cada una de las ejecuciones realizadas es validada y en caso de que el shell mande un estatus final de error, la secuencia es detenida con un estatus de error.

Este job Sequence es ejecutado por medio de un shell job1000x.sh, visto en la sección anterior el cual obtiene el estado final de la ejecución de dicho job, esto es muy importante ya que gracias a ese estatus el shell termina con éxito o con error.

Carga a las tablas de Staging

Esta etapa es la que inicia la carga de la información de los sistemas transaccionales en el DWH, ya que las etapas anteriores eran para preparar la ejecución del proceso de

carga, la cual no es información de estos sistemas y los procesos de extracción sólo crean la información que posteriormente será cargada.

Debido a que los procesos de extracción solo cargan el resultado de su ejecución en la tabla de bitácora, se reutilizaron los usuarios de carga a Staging para hacer esta actividad, ya que no tenía sentido el crear usuarios que sólo actualizaran una tabla de bitácora. Esta etapa al igual que la anterior se usan shells para ejecutar estos procesos, los cuales están representados en la figura 17.

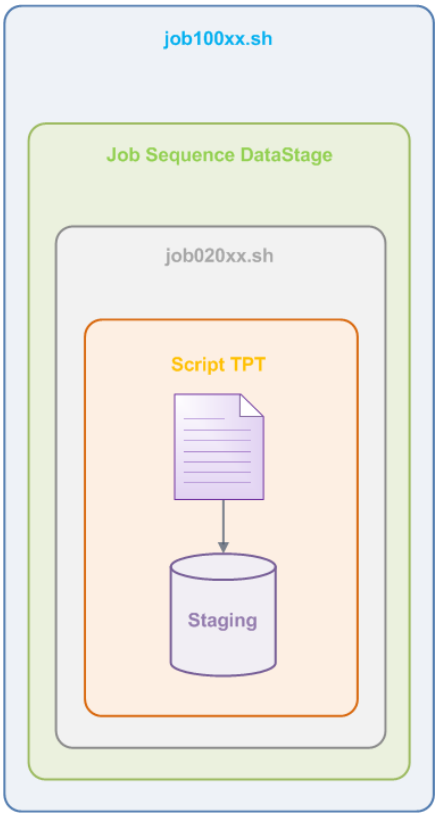


Figura 17. Proceso de carga a Staging (Diseño propio)

A continuación, se explica cómo funciona cada una de las partes que componen este proceso, como ya se ha dicho en secciones anteriores, los shells que ejecutan los procesos de cualquier etapa de carga en el DWH son los job1000x.sh.

Se puede observar que la estructura entre este proceso y el anterior es muy similar, esto lo diseñé pensando en que aún cuando se iban a construir distintas etapas de carga, los procesos no tuvieran que cambiar mucho y fuera muy fácil de asimilar entre cada etapa de carga al DWH. Por lo que estos shells siempre ejecutan jobs Sequence, lo que va cambiando son los shells que cargan cada etapa, en este caso son shells job0200x.sh que ejecutan cada proceso de carga a las tablas de Staging.

En este caso se le conoce como tablas de Staging, a todas las tablas que contienen la información de cada una de las interfaces de carga, ya sean las interfaces generadas por los procesos batch de cada sistema fuente o por los procesos de extracción, como el descrito en la sección anterior.

En estas tablas hay muy pocas validaciones, que se hacen para realizar la carga a las mismas, por lo que la información cargada en estas tablas es prácticamente la misma que la de las interfaces, teniendo sólo las siguientes validaciones

- Columnas de tipo numérico sin caracteres alfanuméricos
- Columnas de tipo fecha con formato de fecha válido
- Que la interfaz cumpla con el número de columnas definido
- Que las columnas no sobrepasen la longitud definida como máxima
- Que el nombre de cada interfaz sea el mismo con el que fue definido
- Que el carácter definido como separador de columna sea el que las interfaces tengan

Estas validaciones se hacen en cada uno de los shells de carga a las tablas de Staging y en el script que carga las interfaces a las tablas, ejecutado en estos shells. El script con el que se hace la carga a las tablas se llama Teradata Parallel Transporter o mejor conocido como TPT. A continuación, se describe el funcionamiento de estos scripts.

```
USING CHARACTER SET ASCII
DEFINE JOB "{TBLPRFXDEST1}"
DESCRIPTION 'File to table'
(
```

```
DEFINE OPERATOR "W_1_o_{TBLPRFXDEST1}"
TYPE LOAD
SCHEMA *
ATTRIBUTES
(
      VARCHAR "UserName",
      VARCHAR "UserPassword",
      VARCHAR "LogTable",
      VARCHAR "TargetTable",
      INTEGER "BufferSize",
      INTEGER "ErrorLimit",
      INTEGER "MaxSessions",
      INTEGER "MinSessions",
      INTEGER "TenacityHours",
      INTEGER "TenacitySleep",
      VARCHAR "AccountID",
      VARCHAR "DateForm",
      VARCHAR "ErrorTable1",
      VARCHAR "ErrorTable2",
      VARCHAR "NotifyExit",
      VARCHAR "NotifyExitIsDLL",
      VARCHAR "NotifyLevel",
      VARCHAR "NotifyMethod",
      VARCHAR "NotifyString",
      VARCHAR "PauseAcq",
      VARCHAR "PrivateLogName",
      VARCHAR "TdpId",
      VARCHAR "TraceLevel",
      VARCHAR "WorkingDatabase"
);

DEFINE OPERATOR OPDDL
TYPE DDL
ATTRIBUTES
(
      VARCHAR TdpId          = '{INF_HOST}',
      VARCHAR UserName       = '{LogonUsr}',
      VARCHAR UserPassword   = '{LogonPsw}',
      VARCHAR AccountID ,
      VARCHAR ErrorList      = '3807',
      INTEGER MaxSessions    = 8
);

DEFINE SCHEMA "W_0_s_{TBLPRFXDEST1}"
(
       "PRODUCTO_CVE"              VARCHAR(30)
      ,"PRODUCTO_DES"              VARCHAR(60)
      ,"FCH_ALTA_PRODUCTO"         VARCHAR(20)
      ,"GPO_ART_PRODUCTO_CVE"      VARCHAR(20)
      ,"PROVEEDOR_CVE"             VARCHAR(30)
      ,"IMPORTE_COSTO_PRODUCTO"    VARCHAR(30)
      ,"IMPORTE_VENTA_PRODUCTO"    VARCHAR(30)
      ,"TALLA"                     VARCHAR(30)
      ,"COLOR_CVE"                 VARCHAR(20)
      ,"COLOR_DES"                 VARCHAR(30)
```

```
      ,"MARCA"                        VARCHAR(30)
      ,"CLASE_CVE"                    VARCHAR(10)
);

DEFINE OPERATOR "W_0_o_{TBLPRFXDEST1}"
TYPE DATACONNECTOR PRODUCER
SCHEMA "W_0_s_{TBLPRFXDEST1}"
ATTRIBUTES
(
      VARCHAR "FileName",
      VARCHAR "Format",
      VARCHAR "OpenMode",
      INTEGER "BlockSize",
      INTEGER "BufferSize",
      INTEGER "RetentionPeriod",
      INTEGER "RowsPerInstance",
      INTEGER "SecondarySpace",
      INTEGER "UnitCount",
      INTEGER "VigilElapsedTime",
      INTEGER "VigilWaitTime",
      INTEGER "VolumeCount",
      VARCHAR "AccessModuleName",
      VARCHAR "AccessModuleInitStr",
      VARCHAR "DirectoryPath",
      VARCHAR "ExpirationDate",
      VARCHAR "IndicatorMode",
      VARCHAR "PrimarySpace",
      VARCHAR "PrivateLogName",
      VARCHAR "RecordFormat",
      VARCHAR "RecordLength",
      VARCHAR "SpaceUnit",
      VARCHAR "TextDelimiter",
      VARCHAR "VigilNoticeFileName",
      VARCHAR "VigilStartTime",
      VARCHAR "VigilStopTime",
      VARCHAR "VolSerNumber",
      VARCHAR "UnitType"
);

STEP A_DDL
(
      APPLY
            ('DROP TABLE {DBPREFIX}.{TBLPRFXDEST1};'),
            ('DROP TABLE {DBWRKPREFIX}.{TBLPRFXDEST1}_ET;'),
            ('DROP TABLE {DBWRKPREFIX}.{TBLPRFXDEST1}_UV;'),
            ('CREATE MULTISET TABLE {DBPREFIX}.{TBLPRFXDEST1} ,NO
FALLBACK ,
                  NO BEFORE JOURNAL,
                  NO AFTER JOURNAL,
                  CHECKSUM = DEFAULT,
                  DEFAULT MERGEBLOCKRATIO
                  (PRODUCTO_CVE DECIMAL(15,0),
                  PRODUCTO_DES VARCHAR(60) CHARACTER,
                  FCH_ALTA_PRODUCTO DATE FORMAT ''YYYY-MM-DD'',
                  GPO_ART_PRODUCTO_CVE SMALLINT,
```

```
                  PROVEEDOR_CVE DECIMAL(15,0),
                  IMPORTE_COSTO_PRODUCTO DECIMAL(18,3),
                  IMPORTE_VENTA_PRODUCTO DECIMAL(18,3),
                  TALLA VARCHAR(30) CHARACTER,
                  COLOR_CVE SMALLINT,
                  COLOR_DES VARCHAR(30) CHARACTER,
                  MARCA VARCHAR(30) CHARACTER,
                  CLASE_CVE SMALLINT)
            PRIMARY INDEX ( SKU_CVE );')
      TO OPERATOR (OPDDL);
);

STEP B_LOAD
(
      APPLY
      (
            'INSERT INTO {DBPREFIX}.{TBLPRFXDEST1}
            (
                   PRPODUCTO_CVE
                  ,PRODUCTO_DES
                  ,FCH_ALTA_PRODUCTO
                  ,GPO_ART_PRODUCTO_CVE
                  ,PROVEEDOR_CVE
                  ,IMPORTE_COSTO_PRODUCTO
                  ,IMPORTE_VENTA_PRODUCTO
                  ,TALLA
                  ,COLOR_CVE
                  ,COLOR_DES
                  ,MARCA
                  ,CLASE_CVE
            )
            VALUES
            (
                   :PRODUCTO_CVE
                  ,:PRODUCTO_DES
                  ,:FCH_ALTA_PRODUCTO
                  ,:GPO_ART_PRODUCTO_CVE
                  ,:PROVEEDOR_CVE
                  ,:IMPORTE_COSTO_PRODUCTO
                  ,:IMPORTE_VENTA_PRODUCTO
                  ,:TALLA
                  ,:COLOR_CVE
                  ,:COLOR_DES
                  ,:MARCA
                  ,:CLASE_CVE
            );'
      )
      TO OPERATOR
      (
            "W_1_o_{TBLPRFXDEST1}"[1]
            ATTRIBUTES
            (
                  "UserName" = '{LogonUsr}',
                  "UserPassword" = '{LogonPsw}',
                  "LogTable" = '{DBWRKPREFIX}.{TBLPRFXDEST1}_log',
```

```
                        "ErrorTable1" = '{DBWRKPREFIX}.{TBLPRFXDEST1}_ET',
                        "ErrorTable2" = '{DBWRKPREFIX}.{TBLPRFXDEST1}_UV',
                        "TargetTable" = '{DBPREFIX}.{TBLPRFXDEST1}',
                        "TdpId" = '{INF_HOST}',
                        "MaxSessions" = 8
                  )
            )
            SELECT * FROM OPERATOR
            (
                  "W_0_o_{TBLPRFXDEST1}"[1]
                  ATTRIBUTES
                  (
                        "FileName" = '{ARCHIVO_CARGA}',
                        "Format" = 'DELIMITED',
                        "OpenMode" = 'Read',
                        "DirectoryPath" =
'/datastage_data/empresa_A/DWH/input/{INF_FUENTE}',
                        "IndicatorMode" = 'N',
                        "TextDelimiter" = '^'
                  )
            );
      );
);
```

En el script se observa que las primeras líneas indican el tipo de set de caracteres con los que se va a trabajar el archivo, en este caso es ASCII, así como el nombre del proceso, el cual para un mayor control tiene el mismo nombre de la tabla a cargar, este valor es enviado por medio del shell a ejecutar, todo lo que se encuentre entre corchetes es un parámetro.

En seguida se definen los parámetros de lectura del TPT, estos valores son necesarios para que el script pueda ejecutarse, así como los parámetros de conexión de la base de datos a la cual vamos a cargar la información. Uno de los parámetros que hace que este tipo de script sea muy eficiente es el de MaxSessions, ya que se le indica cuántas sesiones de paralelismo tendrá la carga de la información.

Los siguientes parámetros que se requieren son los de la tabla a cargar, primero definimos las columnas que se van a leer, después se definen los valores propios de carga del tpt para que el script pueda ejecutarse, debido a que se está haciendo una carga de una fuente externa, en este caso un archivo físico a una tabla, si el proceso llega a fallar la tabla a cargar queda bloqueada, es por esto que se pone dentro del

script que la tabla se borre y se vuelva a crear, con esto evitamos que la tabla se tenga que generar de forma manual si es que se quedó bloqueada por una ejecución previa.

En seguida se tiene el proceso de lectura de las columnas del archivo que estarán llenando la tabla destino, en seguida van los parámetros de la tabla a cargar, aquí se hace una mención especial, ya que cuando se hace una carga de este tipo en Teradata, mientras el proceso está haciendo la carga se crean dos tablas de error una con terminación ET, en donde se almacenan los errores de carga del proceso y la otra con terminación UV, que es donde se almacenan posibles registros duplicados.

Una vez que el proceso termina de ejecutarse, si la carga es correcta se borran las tablas antes mencionadas de forma automática. Estas tablas tienen mucha utilidad, ya que ayudan mucho al momento de revisar los errores de carga, ya que en el caso de la tabla de errores ET, te indica la columna y el tipo de error de carga, en cuanto a la tabla UV, ayuda mucho porque se pueden ver posibles registros duplicados y en un archivo con millones de registros sería muy difícil poderlos identificar.

Finalmente, los últimos parámetros que se tienen son los del archivo de carga, este tipo de script es de los que más cambian entre cada proceso de carga, debido a que viene la descripción de la tabla a cargar, por lo que siempre será diferente en todo lo concerniente a la estructura y/o carga de la tabla.

Este script es ejecutado por un shell job0200x.sh, el cual le manda los parámetros necesarios de ejecución al tpt, que los obtiene del archivo de configuración revisado en la sección de carga inicial del DWH. A continuación, se describe parte del código de este tipo de shell.

```
NU_JOB=job02423
RUTA_LOG=$PATH_LOGS
ARCHIVO_CONFIG=arch_usuarios.ctl
ARCHIVO_PARAM=arch_procesos.ctl
NUM_JOB=`echo $NU_JOB | cut -c4-8`

# Obtiene parametros
INF_FASE=stg
```

```
INF_FUENTE=`grep "$NU_JOB" ${PATH_TMP}${ARCHIVO_PARAM}|tr '[' ' '|tr ']' ' '
|cut -d"^" -f4 | head -1`
INF_USR=`grep "$INF_FASE^$INF_FUENTE" ${PATH_TMP}${ARCHIVO_CONFIG}|tr '[' '
'|tr ']' ' '|cut -d"^" -f1`
INF_PASS=`grep "$INF_FASE^$INF_FUENTE" ${PATH_TMP}${ARCHIVO_CONFIG}|tr '[' '
'|tr ']' ' '|cut -d"^" -f2`
ARCHIVO_DAT=`grep "$NU_JOB^$INF_FASE^$INF_FUENTE" ${PATH_TMP}${ARCHIVO_PARAM}
|tr '[' ' '|tr ']' ' '|cut -d"^" -f7 |head -1`
ARCHIVO_CIF=`grep "$NU_JOB^$INF_FASE^$INF_FUENTE" ${PATH_TMP}${ARCHIVO_PARAM}
|tr '[' ' '|tr ']' ' '|cut -d"^" -f8 | head -1`
NOM_VISTA=`grep "$NU_JOB^$INF_FASE^$INF_FUENTE"  ${PATH_TMP}${ARCHIVO_PARAM}
|tr '[' ' '|tr ']' ' '|cut -d"^" -f10| head -1`

# Validacion de archivo de cifras
validacion_cif()
{
      echo "                                                                "
      echo "________________________________________________________________"
      echo "                                                                "
      echo "            Validacion de cifras ...                            "
      echo "________________________________________________________________"
      echo "                                                                "

      echo "*--- Validando conteo de cifras... ---*"
      CONTEOFILE=`wc -l < ${PATH_INPUT}${INF_FUENTE}/${ARCHIVO_DAT} | sed 's/^
*//' `
      CONTEOCIF=`grep "" ${PATH_INPUT}${INF_FUENTE}/${ARCHIVO_CIF} |tr '[' '
'|tr ']' ' '|cut -d"^" -f3 | sed 's/0*//' | sed 's/[^a-z|0-9]//g;' `
      if  [ "$CONTEOCIF" = "" ];then
            CONTEOCIF=0
      fi
      echo " Los registros de carga del archivo TXT son: ${CONTEOFILE}"
      echo " Los registros de cifras del archivo CIF son: ${CONTEOCIF}"
      if [ ${CONTEOFILE} = "$CONTEOCIF" ]; then
            echo "*--- El numero de registros coincide ---*"
            echo "*--- Se continua con el proceso de carga ---*"
      else
            echo "Los registros leídos e insertados no coinciden para el
archivo ${PATH_INPUT}${INF_FUENTE}/${ARCHIVO_DAT}"
            disp_BTQ_sp "$NUM_JOB, '$NU_JOB', '$INF_FASE', NULL, 1,
'$NOM_VISTA', '$FECHA_PROCESO', 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
'$RUTA_LOG', NULL" "ini"
            disp_BTQ_sp "$NUM_JOB,'$NU_JOB', '$INF_FASE', NULL, 1,
'$NOM_VISTA', '$FECHA_PROCESO', 5, $VALIDA, $VALIDA, $VALIDA, $VALIDA,
$VALIDA,'N/A', '$NU_JOB.$FECHA.log', '$RUTA_LOG', NULL" "err"
            echo "Se cancela proceso de carga"
            exit 1
      fi
}

# Validacion de archivos de datos
valida_arch()
{
      ExisteCCR=`ls ${PATH_INPUT}${INF_FUENTE}/${ARCHIVO_DAT}`
      if [ $? -ne 0 ]; then
```

```
            echo "---------ERROR----------"
            echo "No existe el archivo DAT
${PATH_INPUT}${INF_FUENTE}/${ARCHIVO_DAT}"
            disp_BTQ_sp "$NUM_JOB, '$NU_JOB', '$INF_FASE', NULL, 1,
'$NOM_VISTA', '$FECHA_PROCESO', 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
'$RUTA_LOG', NULL" "ini"
            disp_BTQ_sp "$NUM_JOB,'$NU_JOB', '$INF_FASE', NULL, 1,
'$NOM_VISTA', '$FECHA_PROCESO', 4, $VALIDA, $VALIDA, $VALIDA, $VALIDA,
$VALIDA,'N/A', '$NU_JOB.$FECHA.log', '$RUTA_LOG', NULL" "err"
            echo "Se cancela proceso de carga ${NU_JOB}"
            exit 1
      else
            validacion_cif
      fi
}

# Ejecucion de carga de datos
ejec_carga()
{
      # Variables de script tpt
      NU_SECUENCIA=$1
      DBPREFIX=`grep "$NU_SECUENCIA^$NU_JOB^$INF_FASE^$INF_FUENTE"
${PATH_TMP}${ARCHIVO_PARAMETROS} |tr '[' ' '|tr ']' ' '|cut -d"^" -f9`
      TBLPRFXDEST1=`grep "$NU_SECUENCIA^$NU_JOB^$INF_FASE^$INF_FUENTE"
${PATH_TMP}${ARCHIVO_PARAMETROS} |tr '[' ' '|tr ']' ' '|cut -d"^" -f10`
      LogonUsr=$INF_USR
      LogonPsw=$INF_PASS
      ENV_TP_PREFIX=$INF_HOST
      SCRIPT_TPT=`grep "$NU_SECUENCIA^$NU_JOB^$INF_FASE^$INF_FUENTE"
${PATH_TMP}${ARCHIVO_PARAMETROS} |tr '[' ' '|tr ']' ' '|cut -d"^" -f6`
      SCRIPT_TPT_EJEC=${NU_JOB}_${TBLPRFXDEST1}.tpt
      ARCHIVO_CARGA=`grep "$NU_SECUENCIA^$NU_JOB^$INF_FASE^$INF_FUENTE"
${PATH_TMP}${ARCHIVO_PARAMETROS} |tr '[' ' '|tr ']' ' '|cut -d"^" -f7`

      echo "      Archivo de carga: $ARCHIVO_CARGA"
      ARCHIVO_TPT_LOG=${NU_JOB}_${TBLPRFXDEST1}.${FECHA}.log
      echo "                                                                    "
      echo "____________________________________________________________________"
      echo "                                                                    "
      echo " Comienza carga de datos del proceso ${NU_JOB} del modulo de
cargas "
      echo "____________________________________________________________________"
      echo "                                                                    "

      # Preparando los parametros del script
      sed "s/{LogonUsr}/${LogonUsr}/g; s/{LogonPsw}/${LogonPsw}/g;
s/{TBLPRFXDEST1}/${TBLPRFXDEST1}/g; s/{DBPREFIX}/${DBPREFIX}/g;
      s/{DBWRKPREFIX}/${DBWRKPREFIX}/g; s/{ARCHIVO_CARGA}/${ARCHIVO_CARGA}/g;
s/{INF_HOST}/${INF_HOST}/g; s/{DBCTLPREFIX}/${DBCTLPREFIX}/g;
      s/{INF_FUENTE}/${INF_FUENTE}/g" $PATH_SCRIPTS$SCRIPT_TPT  >
${PATH_SCRIPTS}${SCRIPT_TPT_EJEC}
      chmod 666 ${PATH_SCRIPTS}${SCRIPT_TPT_EJEC}

      # Elimina log anterior
      rm ${PATH_TMP}${NU_JOB}*.out
```

```
        # Ejecuta el script
        tbuild -f ${PATH_SCRIPTS}${SCRIPT_TPT_EJEC} -j ${NU_JOB}
}
```

En la primera línea del código va el proceso que se va a ejecutar, el cual sirve para obtener todos los parámetros necesarios para cargar la tabla en la base de datos de Staging. Para las siguientes validaciones, primero describo cómo es el formato necesario para identificar las interfaces que se usan al cargar en los procesos de Staging. Los procesos pueden usar las siguientes interfaces

- Interfaz de datos. Tiene la información a cargar de los diferentes sistemas fuente, dependiendo de la naturaleza de la información puede ser solo un delta o toda la información.
- Interfaz de cifras. Tiene el número de registros que la interfaz de datos envió, esta interfaz se compone de tres columnas, la primera es el nombre de la interfaz de datos, la segunda es la fecha de información y la tercera es el número de registros que debe tener la interfaz de datos.
- Interfaz de control. Tiene un resumen de la información de la interfaz de datos, generalmente es para interfaces que tienen ventas o información de crédito, esta columna es un conteo o un sumarizado de la información.

Los procesos siempre usan las primeras dos interfaces, para el nombre de las interfaces se compone de la siguiente manera, la primera parte indica si es un catálogo, una dimensión o una tabla de hechos, la segunda parte indica el nombre técnico del sistema fuente, la tercera parte hace alusión a la información que será cargada a nivel negocio, la cuarta parte es la fecha de información a cargar.

Como es un proceso de carga a un DWH, siempre es la información al día anterior, donde la fecha está en formato YYYYMMDD, finalmente es el sufijo de la interfaz, donde el sufijo *dat* es para la interfaz de datos, el *cif* es para la interfaz de cifras y el *ctl* es para la interfaz de control. En el caso de la jerarquía de datos, solo se ocupan las interfaces de datos y de cifras.

Las primeras validaciones que se hacen son, que exista la interfaz de datos y que los registros de la interfaz de cifras sea el mismo que el número de registros de la interfaz de datos, si alguna de estas validaciones falla, el proceso de carga no continua y se detiene en un estado de error.

Al final se hace la carga de la información ejecutando el script de tipo tpt, nuevamente con la ayuda de los parámetros obtenidos del archivo de procesos, por lo que lo único que se cambia en este tipo de shell, es la primera línea donde va el proceso a ejecutar, por lo que con esto se simplifica demasiado la creación de nuevos procesos de carga a Staging. A continuación, se pone el registro del archivo de configuración con el cual se hace la carga para el código analizado

```
job02423,1,stg,fte,,FAC_PRODUCTO.tpt,fac_fte_prod.20190823.dat,fac_fte_prod.2
0190823.cif,STG_FTE,FAC_PRODUCTO,,
```

En el registro los parámetros que toma el proceso son el número de proceso, la etapa del proceso 1 para cargar archivos de datos (como se ve en el ejemplo) y 2 para cargar archivos de control (en este caso no existe ese archivo), la fase del proceso de carga, el nombre del script, el nombre del archivo de datos, el nombre del archivo de cifras y por último el nombre de la base de datos y la tabla donde se va a cargar la información. Analizando el log, que ejecuta este proceso se puede ver como toma los parámetros del registro anterior.

```
 ____________________________________________________________________________
|                                                                            |
|                                                                            |
|                                                                            |
|  ========== ESTE JOB ES EJECUTADO CON     D A T A  S T A G E ==========   |
|                                                                            |
|                                                                            |
|____________________________________________________________________________|
El ambiente es:  Produccion
Comienza ejecución Sat Aug 24 03:21:24 CDT 2019
_____________________________________________________________________________

     Proceso de Ejecucion para JOB: 02423
_____________________________________________________________________________

_____________________________________________________________________________

            Validacion de cifras ...
_____________________________________________________________________________
```

```
*--- Validando conteo de cifras... ---*
 Los registros de carga del archivo TXT son: 321490
 Los registros de cifras del archivo CIF son: 321490
*--- El numero de registros coincide ---*
*--- Se continua con el proceso de carga ---*

Llamado a ejecucion del proceso de carga job02423 ...
_____________________________________________________________________________

                  Ejecucion de Secuencia: 1
_____________________________________________________________________________

    Registrando ejecucion en bitacora ini ...
_____________________________________________________________________________

*--- Finaliza Registro de ejecucion en Bitacora ini correcta  ---*

      Archivo de carga: fac_fte_producto.20190823.dat
_____________________________________________________________________________

 Comienza carga de datos del proceso job02423 del modulo de cargas
_____________________________________________________________________________

Teradata Parallel Transporter Version 14.10.00.10
Job log: /datastage_data/empresa_A/DWH/tmp/job02423-324134.out
Job id is job02423-324134, running on mtydstageeng-pro
Teradata Parallel Transporter SQL DDL Operator Version 14.10.00.10
OPDDL: private log not specified
OPDDL: connecting sessions
OPDDL: The job will use its internal retryable error codes
OPDDL: sending SQL requests
OPDDL: TPT10508: RDBMS error 3807: Object 'STG_FTE.FAC_PRODUCTO_ET' does not
exist.
OPDDL: TPT18046: Warning: error is ignored as requested in ErrorList
OPDDL: TPT10508: RDBMS error 3807: Object 'STG_FTE.FAC_PRODUCTO_UV' does not
exist.
OPDDL: TPT18046: Warning: error is ignored as requested in ErrorList
OPDDL: disconnecting sessions
OPDDL: Total processor time used = '0.02 Second(s)'
OPDDL: Start : Sat Aug 24 03:21:26 2019
OPDDL: End   : Sat Aug 24 03:21:28 2019
Job step A_DDL completed successfully
Teradata Parallel Transporter DataConnector Operator Version 14.10.00.10
Teradata Parallel Transporter Load Operator Version 14.10.00.10
W_1_o_DIM_PROD: private log not specified
W_0_o_DIM_PROD: Instance 1 directing private log report to 'dtacop-dsoper-
9505-1'.
W_0_o_DIM_PROD: DataConnector Producer operator Instances: 1
W_0_o_DIM_PROD: ECI operator ID: 'W_0_o_FAC_PRODUCTO-9505'
W_0_o_DIM_PROD: Operator instance 1 processing file
'/datastage_data/empresa_A/DWH/input/producto/fac_fte_producto.20190823.dat'.
W_1_o_DIM_PROD: connecting sessions
W_1_o_DIM_PROD: The job will use its internal retryable error codes
W_1_o_DIM_PROD: preparing target table
W_1_o_DIM_PROD: entering Acquisition Phase
```

```
W_1_o_DIM_PROD: entering Application Phase
W_1_o_DIM_PROD: Statistics for Target Table:  'STG_FTE.FAC_PRODUCTO'
W_1_o_DIM_PROD: Total Rows Sent To RDBMS:      321490
W_1_o_DIM_PROD: Total Rows Applied:            321490
W_1_o_DIM_PROD: Total Rows in Error Table 1:   0
W_1_o_DIM_PROD: Total Rows in Error Table 2:   0
W_1_o_DIM_PROD: Total Duplicate Rows:          0
W_1_o_DIM_PROD: disconnecting sessions
W_0_o_DIM_PROD: Total files processed: 1.
W_1_o_DIM_PROD: Total processor time used = '0.22 Second(s)'
W_1_o_DIM_PROD: Start : Sat Aug 24 03:21:31 2019
W_1_o_DIM_PROD: End   : Sat Aug 24 03:21:50 2019
Job step B_LOAD completed successfully
Job job01099 completed successfully
Job start: Sat Aug 24 03:21:24 2019
Job end:   Sat Aug 24 03:21:50 2019
_____________________________________________________________________________

           Validacion de errores ...
_____________________________________________________________________________

*--- Ejecución de la carga Exitosa ---*
_____________________________________________________________________________

        Los registros Leidos son:            321490
        Los registros Insertados son:              321490
        Los registros Duplicados son:  0
        Los registros Borrados son:    0
_____________________________________________________________________________

*--- Consulta Finalizada ---*
_____________________________________________________________________________

    Registrando ejecucion en bitacora fin ...
_____________________________________________________________________________

*--- Finaliza Registro de ejecucion en Bitacora fin correcta  ---*
Finaliza ejecución Sat Aug 24 03:22:02 CDT 2019
```

En esta etapa de carga, los posibles mensajes de error que pueden ser cargados en la tabla de bitácora una vez que el proceso ha terminado, son los siguientes:

- Error no existe archivo de Carga. Este error es cuando el archivo de datos no existe.
- Error diferencia en archivo de Cifras. Este error es cuando el archivo de cifras y el archivo de datos no tienen la misma cantidad de registros.
- Error diferencia en registros. Este error es cuando los registros cargados y los que hay en la interfaz no son los mismos.

- Error en la Carga. Este error es cuando hubo algún problema en la carga, como puede ser un error de formato en una columna, que lleguen columnas diferentes a las que se espera en la tabla o por overflow.

En la figura 18 está el modelo de las tablas que se usó para la carga a Staging de la jerarquía de productos.

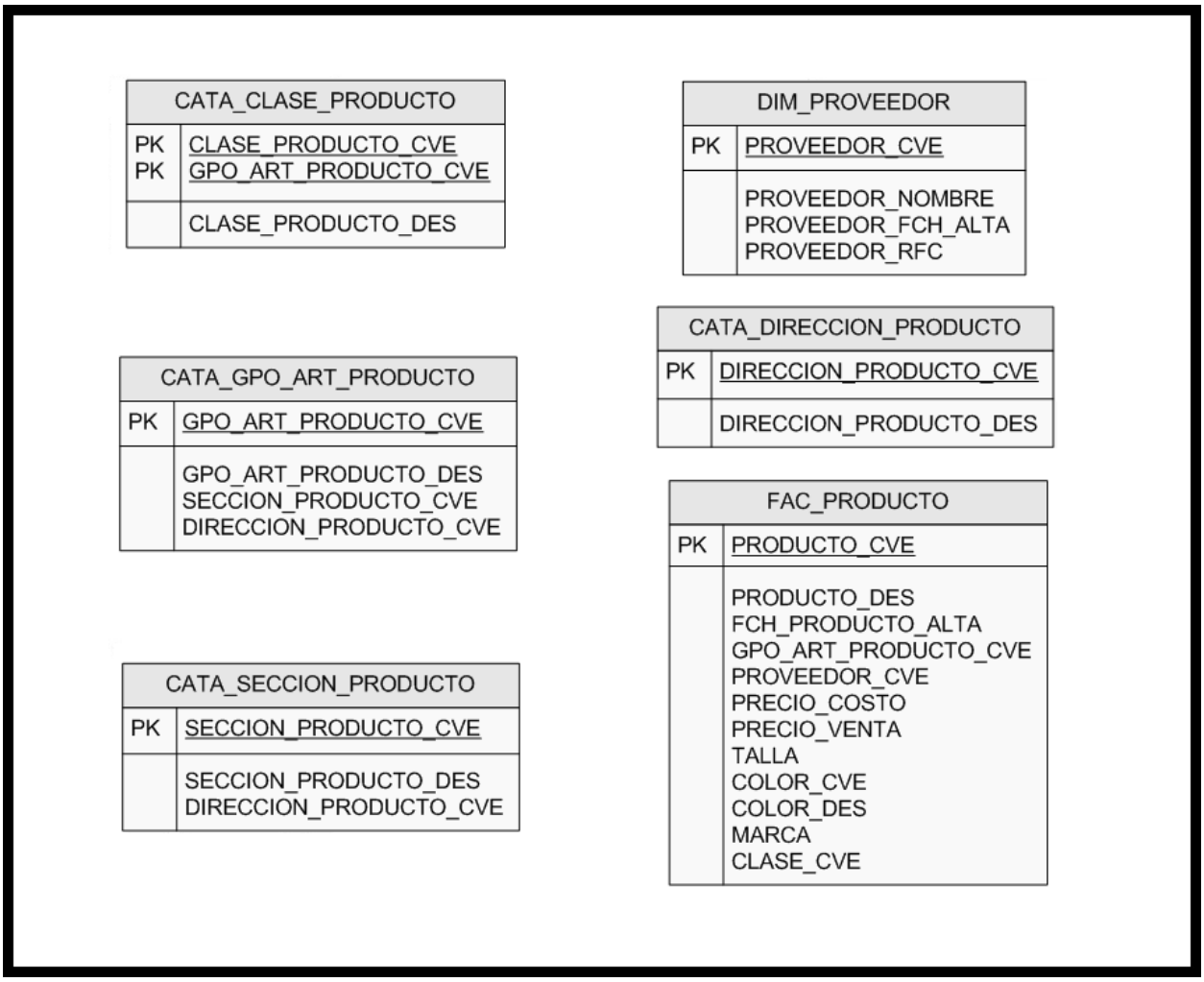


Figura 18. Tablas de Staging de la jerarquía de productos (Diseño propio)

En este modelo no se asigna ninguna relación, debido a que la base de datos ocupada es una de Staging, la cual sólo es la información cargada de las interfaces de los sistemas fuentes.

Los shells de carga a Staging se ejecutan por medio de un job de DataStage de tipo Sequence, el cual sirve para realizar una malla interna de los procesos. Este job es igual que el que se explicó en la sección anterior, por lo que no es necesario volver a explicar esta secuencia. Finalmente, como se ha visto en las otras etapas de carga, este job es ejecutado por medio de un shell job1000x.sh, que también ha sido explicado anteriormente su funcionamiento.

Carga a las tablas de Homologación, Imagen y DWH

Esta etapa es la que finalmente carga la información de los sistemas fuente a las tablas del DWH, este proceso hace la carga de las tablas de Homologación, Imagen (DWI) y DWH, como se muestra en la figura 19.

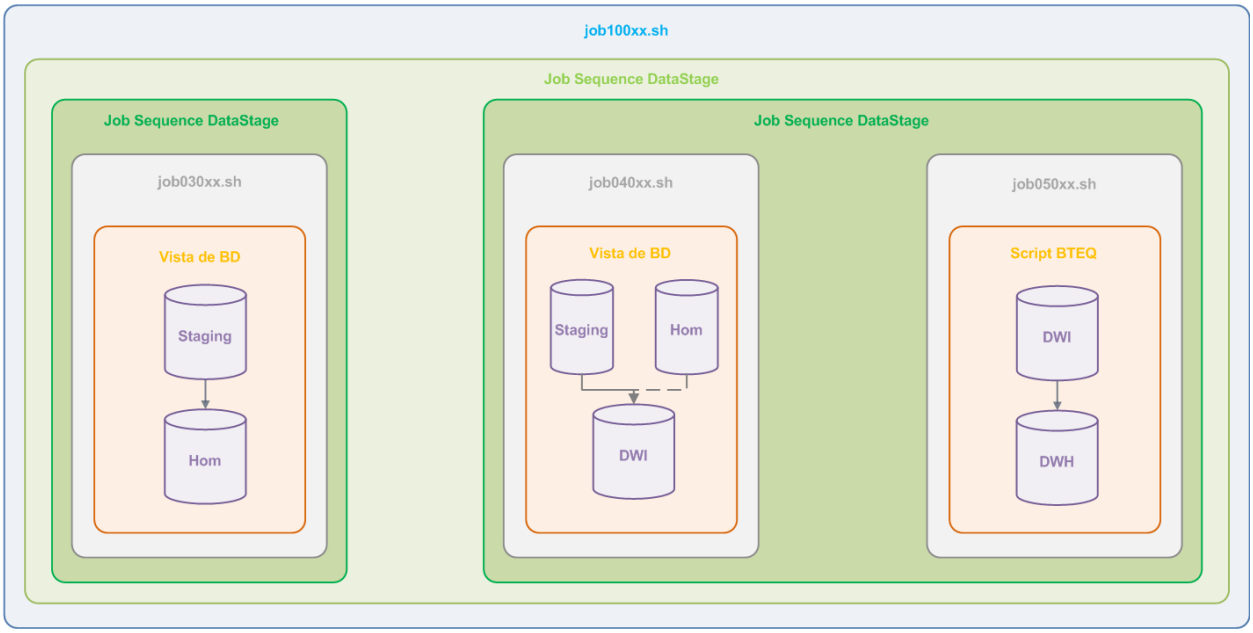


Figura 19. Proceso de carga a DWH (Diseño propio)

A continuación, se explica cómo funciona cada una de las partes que componen este proceso, así como el por qué se ejecutan tres etapas de carga en un proceso y no se hace por separado, finalmente el shell que ejecuta todo el proceso es un job1000x.sh, como ya se ha comentado en las anteriores fases de carga.

Proceso de carga de Homologación

Este proceso no se tenía contemplado originalmente, pero se tuvo que implementar debido a que los datos proporcionados por los sistemas fuentes en algunos casos no tenían claves y sólo eran descripciones. Aunque el caso más extremo sucedió con los datos de la jerarquía de productos que se ha estado viendo a lo largo de las etapas de carga, ya que para el caso de las secciones las claves se repetían.

La jerarquía de productos tiene las siguientes reglas de negocio proporcionadas por la empresa de Retail, la regla indica que los productos que venden están catalogados dentro de un conjunto llamado grupo de artículos, estos grupos de artículos están catalogados dentro de un conjunto llamado sección, estas secciones están catalogadas dentro de un conjunto llamado dirección y al final estas direcciones están catalogadas dentro de la empresa de Retail.

Cada uno de estos niveles de jerarquía tiene una clave única, pero en el caso de las secciones no ocurría de esta manera, ya que cada dirección tiene las mismas claves de secciones, pero con diferente nombre como se observa en la tabla 5.

PROD_CVE	PROD_DES	G_CVE	GPO_DES	S_CVE	SECCION_DES	D_CVE	DIREC_DES
1	LAPTOP A	1	LAPTOPS	1	COMPUTACION	1	ELECTRONICA
2	LAPTOP D	1	LAPTOPS	1	COMPUTACION	1	ELECTRONICA
3	CUCHARAS	2	CUBIERTOS	1	COCINA	2	HOGAR
4	CUCHILLOS	2	CUBIERTOS	1	COCINA	2	HOGAR
6	LICUADORA	3	APARATO COCINA	1	COCINA	2	HOGAR
7	HORNO	3	APARATO COCINA	1	COCINA	2	HOGAR
8	COPA	4	CRISTALERIA	2	COMEDOR	2	HOGAR
9	VASO	4	CRISTALERIA	2	COMEDOR	2	HOGAR

Tabla 5. Jerarquía de productos (Diseño propio)

En la tabla 5 se puede ver el problema que se tenía con las secciones, ya que las secciones de *Cocina* y *Computación* tienen la misma clave, debido a que este modelo se diseñó con catálogos para cada uno de los atributos de la jerarquía, siendo las claves de cada atributo su llave, se tuvo que recurrir a la creación de llaves subrogadas[7], para que el catálogo se pudiera cargar sin tener problemas con las llaves primarias definidas. Este proceso se ejecuta con vistas a nivel de base de datos que se llamaron vistas de transformación de homologación, el código de la vista de las secciones es el siguiente

```
REPLACE VIEW VISTAS_HOM.VCATA_SECCION_SUBROGADA
(
 SECCION_PRODUCTO_SUBROGADA
,SECCION_PRODUCTO_CVE
,DIRECCION_PRODUCTO_CVE
)
AS
SELECT
       CAST((RANK() OVER ( ORDER BY STG.SECCION_PRODUCTO_CVE,
STG.DIRECCION_PRODUCTO_CVE )) +
       (SELECT CASE WHEN MAX(SECCION_PRODUCTO_SUBROGADA) IS NULL THEN 0
                    ELSE MAX(CAST(SECCION_PRODUCTO_SUBROGADA AS INTEGER)) END
AS SECCION_PRODUCTO_SUBROGADA
        FROM HOM.CATA_SECCION_SUBROGADA) AS INTEGER) AS
SECCION_PRODUCTO_SUBROGADA
      ,STG.SECCION_PRODUCTO_CVE
      ,STG.DIRECCION_PRODUCTO_CVE
FROM  (
      SELECT SECCION_PRODUCTO_CVE
            ,SECCION_PRODUCTO_DES
            ,DIRECCION_PRODUCTO_CVE
      FROM  STG_FTE.CATA_SECCION_PRODUCTO
      GROUP BY 1, 2, 3
      ) STG
LEFT JOIN HOM.CATA_SECCION_SUBROGADA HOM
ON    STG.SECCION_PRODUCTO_CVE   = HOM.SECCION_PRODUCTO_CVE
AND   STG.DIRECCION_PRODUCTO_CVE = HOM.DIRECCION_PRODUCTO_CVE
WHERE HOM.SECCION_PRODUCTO_CVE IS NULL
;
```

Para este proceso se creó una tabla de homologación, donde se guarda la clave subrogada que se van generando en este caso HOM.CATA_SECCION_SUBROGADA, en donde se hace un subquery agrupado por las secciones y direcciones para tener datos únicos de la tabla de Staging, este subquery se cruza por la tabla de homologación por la sección y dirección, finalmente en el select se hace una secuencia

[7] Llave creada para tener una clave única en una tabla. Ver anexo

numérica para obtener la llave subrogada ordenando por las claves de la sección y la dirección. En la tabla 6 se observa las nuevas claves de las secciones subrogadas.

PROD_CVE	PROD_DES	G_CVE	GPO_DES	S_CVE	SECCION_DES	D_CVE	DIREC_DES
1	LAPTOP A	1	LAPTOPS	1	COMPUTACION	1	ELECTRONICA
2	LAPTOP D	1	LAPTOPS	1	COMPUTACION	1	ELECTRONICA
3	CUCHARAS	2	CUBIERTOS	2	COCINA	2	HOGAR
4	CUCHILLOS	2	CUBIERTOS	2	COCINA	2	HOGAR
6	LICUADORA	3	APARATO COCINA	2	COCINA	2	HOGAR
7	HORNO	3	APARATO COCINA	2	COCINA	2	HOGAR
8	COPA	4	CRISTALERIA	3	COMEDOR	2	HOGAR
9	VASO	4	CRISTALERIA	3	COMEDOR	2	HOGAR

Tabla 6. Jerarquía de productos con llave subrogada (Diseño propio)

En la tabla 6 se puede ver como la clave de las secciones de *Cocina* y *Computación* ya tienen diferente clave. Esta vista es ejecutada por un shell job0300x.sh, el cual manda los parámetros necesarios para la ejecución de la vista. A continuación, se describe parte de código de este tipo de shell.

```
NU_JOB=job03129
RUTA_LOG=$PATH_LOGS
ARCHIVO_CONFIG=arch_usuarios.ctl
ARCHIVO_PARAMETROS=arch_procesos.ctl
NUM_JOB=`echo $NU_JOB | cut -c4-8`
#Obtiene parametros
INF_FASE=hom
INF_FUENTE=`grep "$NU_JOB" ${PATH_TMP}${ARCHIVO_PARAMETROS} |tr '[' ' '|tr
']' ' '|cut -d"^" -f4 | head -1`
INF_AREA_SUJ=`grep "$NU_JOB" ${PATH_TMP}${ARCHIVO_PARAMETROS} |tr '[' ' '|tr
']' ' '|cut -d"^" -f5 | head -1`
INF_USR=`grep "$INF_FASE^$INF_AREA_SUJ" ${PATH_TMP}${ARCHIVO_CONFIG} |tr '['
' '|tr ']' ' '|cut -d"^" -f1`
INF_PASS=`grep "$INF_FASE^$INF_AREA_SUJ" ${PATH_TMP}${ARCHIVO_CONFIG} |tr '['
' '|tr ']' ' '|cut -d"^" -f2`
DBPREFIXDEST=`grep "$NU_JOB^$INF_FASE^$INF_FUENTE"
${PATH_TMP}${ARCHIVO_PARAMETROS} |tr '[' ' '|tr ']' ' '|cut -d"^" -f9`
DBPREFIXFUEN=`grep "$NU_JOB^$INF_FASE^$INF_FUENTE"
${PATH_TMP}${ARCHIVO_PARAMETROS} |tr '[' ' '|tr ']' ' '|cut -d"^" -f11`
TBLPRFXDEST=`grep "$NU_JOB^$INF_FASE^$INF_FUENTE"
${PATH_TMP}${ARCHIVO_PARAMETROS} |tr '[' ' '|tr ']' ' '|cut -d"^" -f10`
TBLPRFXFUEN=`grep "$NU_JOB^$INF_FASE^$INF_FUENTE"
${PATH_TMP}${ARCHIVO_PARAMETROS} |tr '[' ' '|tr ']' ' '|cut -d"^" -f12`
```

```
#Ejecucion de carga de datos
disp_HOM()
{
      echo "_____________________________________________________________________"
      echo "                                                                     "
      echo "   Realizando ejecucion de carga a HOM ...                           "
      echo "_____________________________________________________________________"
      echo "                                                                     "

      var_carga=${ENV_TP_PREFIX}/${INF_USR},${INF_PASS}
      var_btq=${PATH_TMP}${NU_JOB}_cargaHOM.btq
      var_log=${PATH_LOGS}${NU_JOB}_cargaHOM.${FECHA}.log

      echo ".set ERROROUT STDOUT;"                                  > $var_btq
      echo ".logon $var_carga"                                     >> $var_btq
      echo ".if errorlevel <> 0 then .quit 20;"                    >> $var_btq
      echo "INSERT INTO ${DBPREFIXDEST}.${TBLPRFXDEST}
              SELECT *FROM ${DBPREFIXFUEN}.${TBLPRFXFUEN};"        >> $var_btq
      echo ".if errorcode <> 0 then .quit errorcode;"              >> $var_btq
      echo .logoff                                                 >> $var_btq
      echo .exit errorcode                                         >> $var_btq

      bteq < $var_btq > $var_log
      salida_bteq=$?

      #Validacion de la ejecucion
      if [ "{$salida_bteq}" != "{0}" ] ; then

            DES_ERROR=`grep "*** Failure" $var_log |tail -1|tr '[' ' '|tr ']'
' '| head -1 |sed "s/'//g"`

            echo "*--- Error en proceso bteq de ejecucion carga  ---*"
            echo "Codigo de error: $salida_bteq | $DES_ERROR"
            echo "========[   Termino proceso de carga .....  ]========"
            echo "`date '+%a %Y-%b-%d %H:%M:%S'`"
            exit 1

      else

            echo "*--- Carga de ejecucion exitosa  ---*"

      fi
}
```

Como en las otras fases la primera línea del código indica el proceso a ejecutarse, el cual sirve para obtener todos los parámetros necesarios para su ejecución. Después se ejecuta la carga de la tabla de homologación, creando una variable con el formato de un script de tipo bteq, el cual es ejecutado enseguida y finalmente con la salida obtenida de la ejecución de ese script ponemos el estatus final del proceso de carga. A

continuación, se pone el registro del archivo de configuración con el cual se hace la carga del código analizado

```
job03129,1,hom,,area_sujeto,,,,HOM,CATA_SECCION_SUBROGADA,VISTAS_HOM,VCATA_SE
CCION_SUBROGADA
```

En el registro los parámetros que toma el proceso son el número de proceso, la etapa del proceso 1 (siempre va este valor), la fase del proceso de carga, el área sujeto a la cual vamos a cargar la información, el nombre de la base de datos y la tabla donde se va a cargar la información y por último el nombre de la base de datos y la tabla origen de la información.

No se debe de confundir el área sujeto del DWH con el sistema fuente, para este caso es el mismo, pero hay otros procesos que cargan a las mismas tablas del DWH, aunque vienen de distintos sistemas fuentes, por ejemplo los clientes. Analizando el log, que ejecuta este proceso se puede ver como toma los parámetros del registro anterior.

```
 _____________________________________________________________________
|                                                                     |
|                                                                     |
|                                                                     |
|  ========== ESTE JOB ES EJECUTADO CON     D A T A  S T A G E ==========  |
|                                                                     |
|                                                                     |
|_____________________________________________________________________|
El ambiente es:  Produccion
Comienza ejecución Sun Aug 25 07:13:32 CDT 2019
_____________________________________________________________________

     Proceso de Ejecucion para JOB: 03129
_____________________________________________________________________

_____________________________________________________________________

    Registrando ejecucion en bitacora ini ...
_____________________________________________________________________

*--- Registro de ejecucion en Bitacora ini correcta  ---*
_____________________________________________________________________

   Realizando ejecucion de carga a HOM ...
_____________________________________________________________________
INSERT INTO HOM.CATA_SECCION_SUBROGADA
            SELECT * FROM VISTAS_HOM.VCATA_SECCION_SUBROGADA;
```

```
 *** Insert completed. 2 rows added.
 *** Total elapsed time was 1 second.


*--- Carga de ejecucion exitosa  ---*
Registros Insertados: 2
Registros Borrados: 0
_____________________________________________________________________________

    Registrando ejecucion en bitacora fin ...
_____________________________________________________________________________

*--- Registro de ejecucion en Bitacora fin correcta  ---*
Finaliza ejecución Sun Aug 25 07:13:36 CDT 2019
```

En esta etapa de carga, el posible mensaje de error que puede ser cargado en la tabla de bitácora una vez que el proceso ha terminado, es el siguiente:

- Error en la Carga. Este error es cuando hubo algún problema en la carga, como puede ser un error de formato en una columna, que lleguen columnas diferentes a las que se espera en la tabla o por overflow.

Los shells de carga a Homologación se ejecutan por medio de un job de DataStage de tipo Sequence, el cual realiza una malla interna de estos procesos, este job ya se ha explicado anteriormente.

Proceso de carga de Imagen y DWH

Los procesos de carga de Imagen y DWH se hacen al mismo tiempo, porque la carga al DWH se hace por el tipo de tabla, es decir primero se cargan los catálogos, después las tablas dimensionales y por último las tablas transaccionales.

Esto lo diseñé de esta manera, porque si un usuario consulta una tabla y hace un cruce con algún catálogo, si hay claves nuevas que están en la tabla y que aún no se han cargado en el catálogo su información quedaría inconsistente, pero si primero se cargan los catálogos ya se tendrán las posibles nuevas claves aún cuando las demás

tablas todavía no se hayan cargado al DWH. Con esto evitamos problemas de integridad de la información, se puede ver en la figura 20 como se diseño la secuencia de carga al DWH.

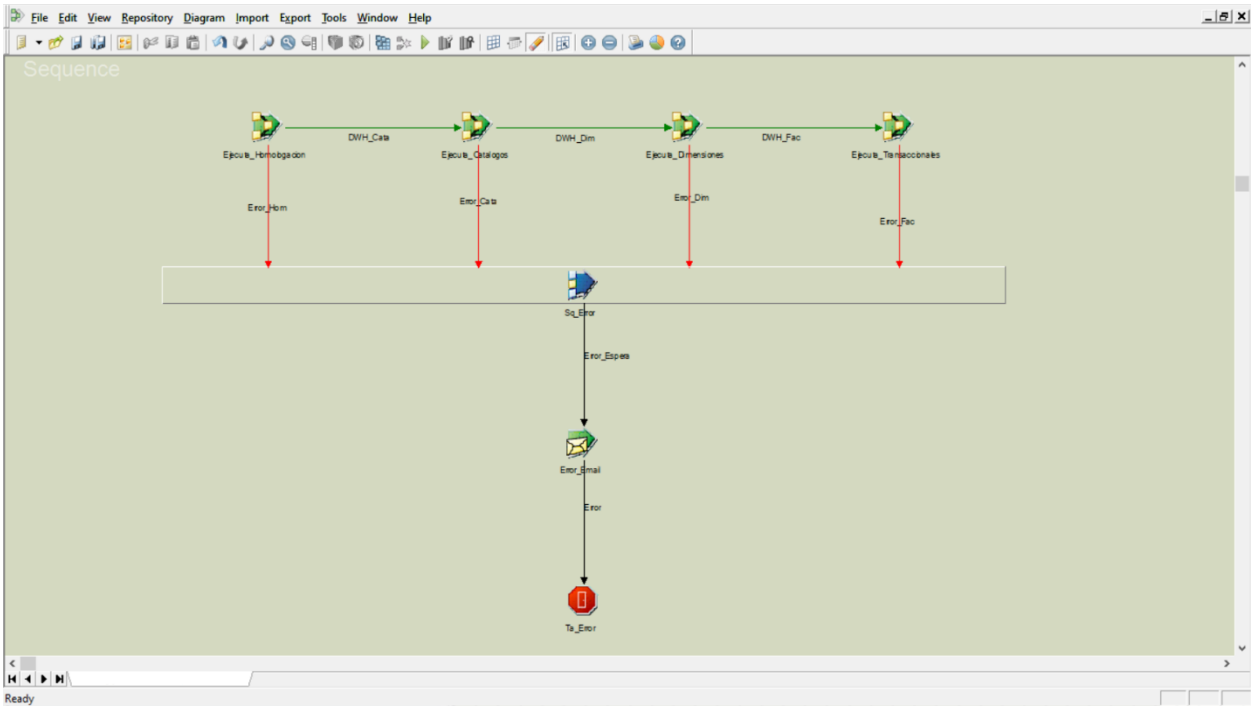


Figura 20. Secuencia de ejecución de carga al DWH (Designer DataStage)

A continuación, se explica cómo se carga el proceso de Imagen también conocido como Data Warehouse Image (DWI), este proceso es muy importante porque la información cargada en estas tablas ya tiene las reglas de negocio, la diferencia es que en esta base de datos sólo está la información del día, en cambio en el DWH está la información histórica.

Debido a que en algunos casos hay varias reglas que se tienen que implementar y su solución no es trivial, esta etapa la diseñé en conjunto con el arquitecto de la base de datos con vistas a nivel de base de datos, ya que al momento de hacer pruebas o correcciones es más fácil en una vista, ya que se puede trabajar en el código mientras la vista sigue funcionando.

A continuación, se revisa el código del proceso de carga de las secciones y de los grupos de artículos, ya que en ambos casos se integra la tabla de Homologación vista en la sección anterior, debido al modelo de la jerarquía de productos. La vista del catálogo de las secciones es el siguiente

```
REPLACE VIEW VISTAS_TRN.VCATA_SECCION_PRODUCTO
(
 SECCION_PRODUCTO_ID
,SECCION_PRODUCTO_CVE
,SECCION_PRODUCTO_DES
,DIRECCION_PRODUCTO_CVE
,FECHA_CARGA
)
AS
SELECT
       STG.SECCION_PRODUCTO_SUBROGADA AS SECCION_PRODUCTO_ID
      ,STG.SECCION_PRODUCTO_CVE
      ,STG.SECCION_PRODUCTO_DES
      ,STG.DIRECCION_PRODUCTO_CVE
      ,STG.FECHA_CARGA
FROM (SELECT
             HOM.SECCION_PRODUCTO_SUBROGADA
            ,SEC.SECCION_PRODUCTO_CVE
            ,SEC.SECCION_PRODUCTO_DES
            ,SEC.DIRECCION_PRODUCTO_CVE
            ,CURRENT_DATE AS FECHA_CARGA
      FROM  STG_FTE.CATA_SECCION_PRODUCTO SEC
      INNER JOIN HOM.CATA_SECCION_SUBROGADA HOM
      ON    SEC.SECCION_PRODUCTO_CVE    = HOM.SECCION_PRODUCTO_CVE
      AND   SEC.DIRECCION_PRODUCTO_CVE = HOM.DIRECCION_PRODUCTO_CVE) STG
LEFT JOIN DWH.CATA_SECCION_PRODUCTO DWH
ON    STG.SECCION_PRODUCTO_SUBROGADA = DWH.SECCION_PRODUCTO_ID
WHERE DWH.SECCION_PRODUCTO_ID IS NULL
OR    STG.SECCION_PRODUCTO_DES <> DWH.SECCION_PRODUCTO_DES
;
```

Al revisar el query anterior, se hace un subquery cruzando la tabla de Staging y la tabla de Homologación analizada en la sección anterior, al final se cruza con la tabla del DWH, para agregar las nuevas secciones o las secciones que cambiaron de nombre. La vista de la tabla de los grupos de artículos es el siguiente

```
REPLACE VIEW VISTAS_TRN.VCATA_GPO_ART_PRODUCTO
(
 GPO_ART_PRODUCTO_CVE
,GPO_ART_PRODUCTO_DES
,SECCION_PRODUCTO_ID
```

```
,FECHA_CARGA
)
AS
SELECT
       STG.GPO_ART_PRODUCTO_CVE
      ,STG.GPO_ART_PRODUCTO_DES
      ,HOM.SECCION_PRODUCTO_SUBROGADA AS SECCION_PRODUCTO_ID
      ,CURRENT_DATE AS FECHA_CARGA
FROM  STG_FTE.CATA_GPO_ART_PRODUCTO STG
LEFT JOIN HOM.CATA_SECCION_SUBROGADA HOM
ON    STG.SECCION_PRODUCTO_CVE    = HOM.SECCION_PRODUCTO_CVE
AND   STG.DIRECCION_PRODUCTO_CVE = HOM.DIRECCION_PRODUCTO_CVE
LEFT JOIN DWH.CATA_GPO_ART_PRODUCTO DWH
ON    STG.GPO_ART_PRODUCTO_CVE = DWH.GPO_ART_PRODUCTO_CVE
WHERE DWH.GPO_ART_PRODUCTO_CVE IS NULL
OR    STG.GPO_ART_PRODUCTO_DES <> DWH.GPO_ART_PRODUCTO_DES
;
```

Al revisar el query anterior, se hace un cruce de la tabla de Staging con la tabla de Homologación de secciones, ya que se necesita poner la clave de la sección subrogada, para cruzar con el catálogo de secciones del DWH, después se cruza con la tabla del DWH, para agregar los nuevos grupos de artículos o los grupos de artículos que cambiaron de nombre. Finalmente, se revisa la vista de la tabla de los productos ya que esta vista no cruza por ninguna tabla de Homologación, el código es el siguiente

```
REPLACE VIEW VISTAS_TRN.VFAC_PRODUCTO
(
 PRODUCTO_CVE
,PRODUCTO_DES
,FECHA_ALTA_PRODUCTO
,GPO_ART_PRODUCTO_CVE
,PROVEEDOR_CVE
,IMPORTE_COSTO_PRODUCTO
,IMPORTE_VENTA_PRODUCTO
,TALLA
,COLOR_PRODUCTO_CVE
,MARCA
,CLASE_PRODUCTO_CVE
,FCH_CARGA
)
AS
SELECT
       STG.PRODUCTO_CVE
      ,STG.PRODUCTO_DES
      ,STG.FECHA_ALTA_PRODUCTO
      ,STG.GPO_ART_PRODUCTO_CVE
      ,STG.PROVEEDOR_CVE
      ,STG.IMPORTE_COSTO_PRODUCTO
      ,STG.IMPORTE_VENTA_PRODUCTO
```

```
      ,STG.TALLA
      ,STG.COLOR_PRODUCTO_CVE
      ,STG.MARCA
      ,STG.CLASE_PRODUCTO_CVE
      ,CURRENT_DATE AS FCH_CARGA
FROM  STG_FTE.FAC_PRODUCTO STG
LEFT JOIN DWH.FAC_PRODUCTO DWH
ON    STG.PRODUCTO_CVE = DWH.PRODUCTO_CVE
WHERE DWH.PRODUCTO_CVE IS NULL
OR    STG.PRODUCTO_DES           <> DWH.PRODUCTO_DES
OR    STG.GPO_ART_PRODUCTO_CVE   <> DWH.GPO_ART_PRODUCTO_CVE
OR    STG.IMPORTE_COSTO_PRODUCTO <> DWH.IMPORTE_COSTO_PRODUCTO
OR    STG.IMPORTE_VENTA_PRODUCTO <> DWH.IMPORTE_VENTA_PRODUCTO
OR    STG.TALLA                  <> DWH.TALLA
OR    STG.MARCA                  <> DWH.MARCA
;
```

Al revisar el query anterior, se hace un cruce de la tabla de Staging con la tabla de DWH de productos, para ir agregando los nuevos productos o los productos que cambiaron de nombre, de grupo de artículo, de importe de costo, de importe de venta, de talla o de marca.

Estas vistas se ejecutan por medio de shells job0400x.sh, al igual que los shells anteriores, se mandan los parámetros necesarios para la ejecución de la vista. Estos shells son prácticamente iguales a los que se vieron anteriormente en la sección de Homologación, por lo que no es necesario ver el código nuevamente, ya que lo único que cambia es el número de proceso a ejecutar y la función disp_HOM se cambia por disp_DWI, al igual que los nombres de los logs generados se cambia en los nombres el HOM por DWI. A continuación, se pone el registro del archivo de configuración con el cual se hace la carga del proceso de productos

```
job04580,1,dwi,,area_sujeto,,,,DWI,FAC_PRODUCTO,VISTAS_TRN,VFAC_PRODUCTO
```

En el registro los parámetros que toma el proceso son el número de proceso, la etapa del proceso 1 (siempre va este valor), el área sujeto a la cual vamos a cargar la información, el nombre de la base de datos y la tabla donde se va a cargar la información y por último el nombre de la base de datos y la tabla origen de la información. Analizando el log, que ejecuta este proceso se puede ver como toma los parámetros del registro anterior.

```
 ____________________________________________________________________________
|                                                                            |
|                                                                            |
|                                                                            |
|  ========== ESTE JOB ES EJECUTADO CON     D A T A  S T A G E ==========    |
|                                                                            |
|                                                                            |
|____________________________________________________________________________|
El ambiente es:  Produccion
Comienza ejecución Sun Aug 25 07:20:56 CDT 2019
____________________________________________________________________________

     Proceso de Ejecucion para JOB: 04580
____________________________________________________________________________


____________________________________________________________________________

    Registrando ejecucion en bitacora ini ...
____________________________________________________________________________

*--- Registro de ejecucion en Bitacora ini correcta  ---*
____________________________________________________________________________

   Realizando ejecucion de carga a DWI ...
____________________________________________________________________________

      INSERT INTO DWI.FAC_PRODUCTO
            SELECT * FROM VISTAS_TRN.VFAC_PRODUCTO;

 *** Insert completed. 1057 rows added.
 *** Total elapsed time was 1 second.


*--- Carga de ejecucion exitosa  ---*
Registros Insertados: 1057
Registros Borrados: 913
____________________________________________________________________________

    Registrando ejecucion en bitacora fin ...
____________________________________________________________________________

*--- Registro de ejecucion en Bitacora fin correcta  ---*
Finaliza ejecución Sun Aug 25 07:20:59 CDT 2019
```

En esta etapa de carga, el posible mensaje de error que pueden ser cargado en la tabla de bitácora una vez que el proceso ha terminado, es el siguiente:

- Error en la Carga. Este error es cuando hubo algún problema en la carga, como puede ser un error de formato en una columna, que lleguen columnas diferentes

a las que se espera en la tabla, por overflow o por llave primaria duplicada.

Para el proceso de la carga a DWH la diseñé con los scripts de tipo bteq, debido a la versatilidad que ya se ha visto anteriormente en la etapa de carga inicial. Esto debido a que en algunos procesos además de la carga al DWH, se deben de hacer algunas actualizaciones en algunos campos, por lo que la forma más fácil de hacerlo es con estos scripts. A continuación, se describe parte del código que carga a la tabla de productos

```
------------------------------------------------------------------------
------------DELETE PARA UPDATE------------------------------------------
------------------------------------------------------------------------
DELETE ${DBPREFIXDEST}.${TBLPRFXDEST}
WHERE  ${DBPREFIXDEST}.${TBLPRFXDEST}.SKU_CVE =
${DBPREFIXFUEN}.${TBLPRFXFUEN}.SKU_CVE
;


------------------------------------------------------------------------
----INSERT DE NUEVOS REGISTROS Y ACTUALIZADOS--------------------------
------------------------------------------------------------------------
INSERT INTO ${DBPREFIXDEST}.${TBLPRFXDEST}
SELECT *
FROM  ${DBPREFIXFUEN}.${TBLPRFXFUEN}
;
```

Debido a lo comentado en el proceso de carga a DWI, en cuanto a la información que existe en las tablas de Imagen y las de DWH, el proceso lo que hace es un delete de la posible información que haya sido actualizada y en seguida, se inserta la información de DWI a la tabla de DWH.

Estos scripts se ejecutan por medio de shells job0500x.sh, al igual que los shells anteriores, se mandan los parámetros necesarios para la ejecución del script bteq, teniendo los cambios ya descritos en la fase anterior, para no crear más parámetros el nombre del script es la unión del número del proceso y la tabla destino, en este caso sería *job05580_FAC_PRODUCTO.btq*. A continuación, se pone el registro del archivo de configuración con el cual se hace la carga del proceso de productos

```
job05580,1,dwh,,area_sujeto,,,,DWH,FAC_PRODUCTO,DWI,FAC_PRODUCTO
```

En el registro, los parámetros que toma el proceso son el número de proceso, la etapa del proceso 1 (siempre va este valor), la fase del proceso de carga, el área sujeto a la cual vamos a cargar la información, el nombre de la base de datos y la tabla donde se va a cargar la información y por último el nombre de la base de datos y la tabla origen de la información. Analizando el log, que ejecuta este proceso se puede ver como toma los parámetros del registro anterior.

```
 _____________________________________________________________________________
|                                                                             |
|                                                                             |
|                                                                             |
|  ========= ESTE JOB ES EJECUTADO CON     D A T A  S T A G E ==========     |
|                                                                             |
|                                                                             |
|_____________________________________________________________________________|
El ambiente es:  Produccion
Comienza ejecución Sun Aug 25 07:21:02 CDT 2019
_______________________________________________________________________________

     Proceso de Ejecucion para JOB: 05580
_______________________________________________________________________________

_______________________________________________________________________________

    Registrando ejecucion en bitacora ini ...
_______________________________________________________________________________

*--- Registro de ejecucion en Bitacora ini correcta  ---*
_______________________________________________________________________________

   Realizando ejecucion de carga a DWH ...
_______________________________________________________________________________

      INSERT INTO DWH.FAC_PRODUCTO
            SELECT * FROM DWI.FAC_PRODUCTO;

 *** Insert completed. 1057 rows added.
 *** Total elapsed time was 2 seconds.


*--- Carga de ejecucion exitosa  ---*
Registros Insertados: 1057
Registros Borrados: 49
_______________________________________________________________________________

    Registrando ejecucion en bitacora fin ...
_______________________________________________________________________________

*--- Registro de ejecucion en Bitacora fin correcta  ---*
Finaliza ejecución Sun Aug 25 07:21:09 CDT 2019
```

En esta etapa de carga, el posible mensaje de error que pueden ser cargado en la tabla de bitácora una vez que el proceso ha terminado, es el siguiente:

- Error en la Carga. Este error es cuando hubo algún problema en la carga, como puede ser un error de formato en una columna, que lleguen columnas diferentes a las que se espera en la tabla, por overflow o por llave primaria duplicada.

Los shells de carga a Imagen y DWH se ejecutan por medio de un job de DataStage de tipo Sequence, el cual realiza una malla interna de estos procesos, este job ya se ha explicado anteriormente. Este job de DataStage y el que ejecuta los shells de las tablas de Homologación, son ejecutados por otro job de DataStage de tipo Sequence explicado en la figura 20 anteriormente. Finalmente, como se ha visto en las otras etapas de carga, el job Sequence es ejecutado por medio de un shell job1000x.sh.

Para el diseño del DWH, el arquitecto de la base de datos y yo consideramos el espacio que las tablas podían ocupar si las tablas transaccionales no tenían catálogos asociados, porque al guardar una clave numérica en lugar de una columna de tipo alfanumérico el espacio podía crecer exponencialmente. Esto lo tomamos en cuenta desde el inicio del proyecto, debido a que en las primeras reuniones se nos dijo que la información que se requería en el DWH era desde el año 2010, en el caso que la información pudiera ser cuantificada por año y mes.

Sin embargo, en el caso de la información guardada de los productos, no puede ser desde un año en específico, ya que aún cuando se tiene una columna con la fecha en que los productos fueron dados de alta, esto no significa que las ventas que se hayan hecho en el año 2010 no existan productos que se tengan en la empresa de Retail con fechas de alta de años anteriores.

Por esta situación, hay tablas que se pudieron acotar por antigüedad y otras se tienen que cargar con toda la información disponible, como es el caso de los productos. A continuación, en la figura 21 se describe el modelo entidad-relación de la jerarquía de productos teniendo en cuenta los puntos anteriores.

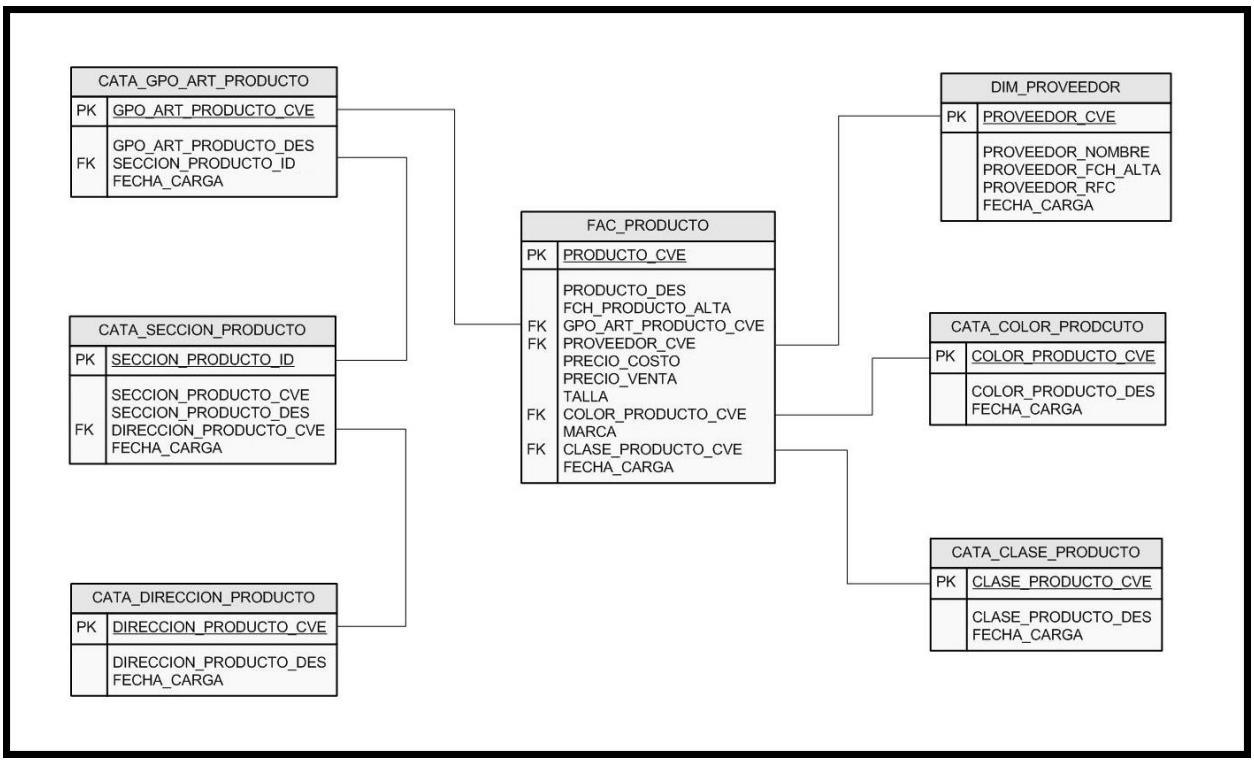


Figura 21. Modelo entidad-relación de la jerarquía de productos en el DWH (Diseño propio)

En el modelo se observa que la llave subrogada es de utilidad para la relación entre las tablas de secciones y de grupo de artículos, pero ese dato no es relevante para el negocio, por lo que se agregan las claves que se tiene desde el sistema fuente. Por último, la tabla de la clase de productos sirve para indicar si un producto es un servicio o un producto físico. Por ejemplo, tomando uno de los productos de la tabla 6 vista anteriormente una licuadora es un producto físico, en cambio el pago de un seguro es un servicio.

Carga a las tablas de Semántica

Esta etapa se creó debido a que la empresa de Retail pidió unas tablas donde los usuarios pudieran hacer consultas para reporteo y minería de datos, con reglas muy específicas de negocio y en algunos casos, se requieren uniones de tablas de distintos sistemas fuentes.

Esta etapa al igual que las anteriores se usan shells para ejecutar estos procesos, los cuales están representados en la figura 22.

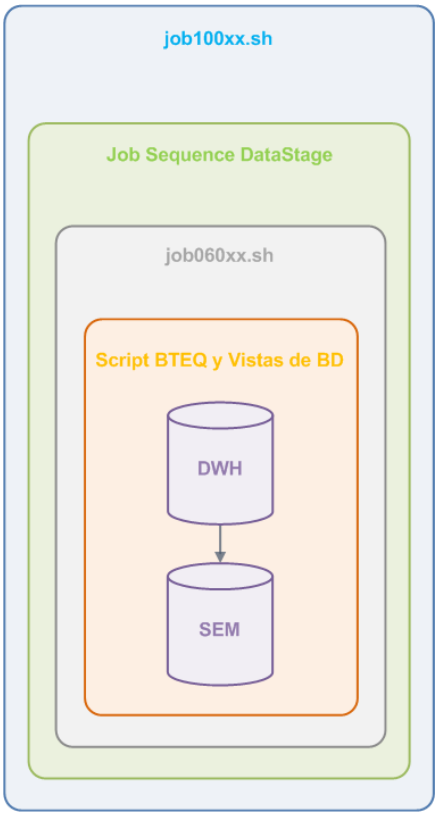


Figura 22. Proceso de carga a Semántica (Diseño propio)

A continuación, se explica cómo funciona cada una de las partes que componen este proceso, como ya se ha dicho en secciones anteriores, los shells que ejecutan los procesos de cualquier etapa de carga en el DWH, incluyendo la Semántica son los job1000x.sh.

Debido a que este proceso es el más complicado, por las reglas de negocio que se necesitan para obtener los datos como se desean, este proceso combina las vistas de transformación y los scripts de tipo bteq. A continuación, se verá el proceso de la creación de la semántica de la jerarquía de productos, el código del script bteq es el siguiente

```
.SET ERROROUT STDOUT;
.LOGON ${var_carga}
.SET WIDTH 1000;
.IF errorlevel <> 0 THEN .quit 20

-- Creacion de tabla temporal de productos
CREATE SET TABLE SEM.FAC_PRODUCTO_TMP ,NO FALLBACK ,
     NO BEFORE JOURNAL,
     NO AFTER JOURNAL,
     CHECKSUM = DEFAULT,
     DEFAULT MERGEBLOCKRATIO
     (
        PRODUCTO_CVE DECIMAL(15,0),
        PRODUCTO_DES VARCHAR(60) CHARACTER,
        FCH_PRODUCTO_ALTA DATE FORMAT 'YYYY-MM-DD',
        PROVEEDOR_CVE DECIMAL(15,0),
        PROVEEDOR_NOMBRE VARCHAR(70) CHARACTER,
        PRECIO_COSTO DECIMAL(18,3),
        PRECIO_VENTA DECIMAL(18,3),
        GPO_ART_PRODUCTO_CVE SMALLINT,
        GPO_ART_PRODUCTO_DES VARCHAR(40) CHARACTER,
        SECCION_PRODUCTO_CVE SMALLINT,
        SECCION_PRODUCTO_DES VARCHAR(40) CHARACTER,
        DIRECCION_PRODUCTO_CVE SMALLINT,
        DIRECCION_PRODUCTO_DES VARCHAR(40) CHARACTER,
        COLOR_PRODUCTO_CVE SMALLINT,
        COLOR_PRODUCTO_DES VARCHAR(40) CHARACTER,
        TALLA VARCHAR(30) CHARACTER,
        CLASE_PRODUCTO_CVE SMALLINT,
        CLASE_PRODUCTO_DES VARCHAR(30) CHARACTER)
PRIMARY INDEX ( PRODUCTO_CVE );

-- Inserta datos en la tabla temporal de productos
INSERT INTO SEM.FAC_PRODUCTO_TMP
SELECT
       PROD.PRODUCTO_CVE
      ,PROD.PRODUCTO_DES
      ,PROD.FCH_PRODUCTO_ALTA
      ,PROD.PROVEEDOR_CVE
      ,PROV.PROVEEDOR_NOMBRE
      ,PROD.PRECIO_COSTO
      ,PROD.PRECIO_VENTA
      ,PROD.GPO_ART_PRODUCTO_CVE
      ,GPO.GPO_ART_PRODUCTO_DES
      ,GPO.SECCION_PRODUCTO_CVE
      ,SEC.SECCION_PRODUCTO_DES
      ,SEC.DIRECCION_PRODUCTO_CVE
      ,DIR.DIRECCION_PRODUCTO_DES
      ,PROD.COLOR_PRODUCTO_CVE
      ,COLOR.COLOR_PRODUCTO_DES
      ,PROD.TALLA
      ,PROD.CLASE_PRODUCTO_CVE
      ,CLASE.CLASE_PRODUCTO_DES
FROM  DWH.FAC_PRODUCTO PROD
LEFT JOIN   DWH.CATA_GPO_ART_PRODUCTO GPO
```

```
ON    PROD.GPO_ART_PRODUCTO_CVE = GPO.GPO_ART_PRODUCTO_CVE
LEFT JOIN   DWH.CATA_SECCION_PRODUCTO SEC
ON    GPO.SECCION_PRODUCTO_ID = SEC.SECCION_PRODUCTO_ID
LEFT JOIN   DWH.CATA_DIRECCION_PRODUCTO DIR
ON    SEC.DIRECCION_PRODUCTO_CVE = DIR.DIRECCION_PRODUCTO_CVE
LEFT JOIN   DWH.CATA_COLOR_PRODUCTO COLOR
ON    PROD.COLOR_PRODUCTO_CVE = COLOR.COLOR_PRODUCTO_CVE
LEFT JOIN   DWH.CATA_CLASE_PRODUCTO CLASE
ON    PROD.CLASE_PRODUCTO_CVE = CLASE.CLASE_PRODUCTO_CVE
LEFT JOIN   DWH.DIM_PROVEEDOR PROV
ON    PROD.PROVEEDOR_CVE = PROV.PROVEEDOR_CVE;

-- Estadisticas en la tabla temporal de productos
COLLECT STATISTICS ON SEM.FAC_PRODUCTO_TMP INDEX ( PRODUCTO_CVE );

-- Delete de dos años de historia en la tabla de semantica
DELETE      ${DBPREFIXDEST}.${TBLPRFXDEST}
WHERE ${DBPREFIXDEST}.${TBLPRFXDEST}.FCH_COMPRA <= SELECT
ADD_MONTHS(FCH_COMPRA, - 24) FROM ${DBPREFIXFUEN}.${TBLPRFXFUEN};

-- Delete de la tabla de semantica para update
DELETE      ${DBPREFIXDEST}.${TBLPRFXDEST}
WHERE ${DBPREFIXDEST}.${TBLPRFXDEST}.FCH_COMPRA =
${DBPREFIXFUEN}.${TBLPRFXFUEN}.FCH_COMPRA;

-- Insert de los nuevos datos a la tabla de semantica
INSERT INTO ${DBPREFIXDEST}.${TBLPRFXDEST}
SELECT *
FROM  ${DBPREFIXFUEN}.${TBLPRFXFUEN};

-- Borra la tabla temporal de productos
DROP TABLE SEM.FAC_PRODUCTO_TMP;
```

En el script se observa que las primeras líneas son los parámetros de conexión a la base de datos, después se crea una tabla temporal con todos los datos necesarios de la jerarquía de productos, enseguida se insertan los datos de la jerarquía de productos, después de la inserción de dichos datos se le crean estadísticas a la tabla temporal para ayudar con los cruces con las otras tablas del DWH.

Para las tablas de la semántica, sólo existirá dos años de historia, por lo tanto se hacen dos *delete*, el primero es para la fecha de compra mayor a dos años y el segundo es por la fecha actual de compra, este último por si es un reproceso de la información, después se insertan los datos en la tabla final de la semántica por medio de una vista lógica, llamada vista de transformación y finalmente se borra la tabla temporal para liberar espacio en la base de datos. A continuación, se pone el código de la vista de transformación

```
REPLACE VIEW VISTAS_SEM.VFAC_VENTAS
(
 TICKET
,MONTO_TICKET
,TIPO_PAGO_TICKET_CVE
,TIPO_PAGO_TICKET_DES
,FCH_TICKET
,TIENDA_CVE
,TIENDA_DES
,PRODUCTO_CVE
,PRODUCTO_DES
,CANTIDAD_PRODUCTO
,MONTO_PRODUCTO
,FCH_PRODUCTO_ALTA
,PROVEEDOR_CVE
,PROVEEDOR_NOMBRE
,PRECIO_COSTO
,PRECIO_VENTA
,GPO_ART_PRODUCTO_CVE
,GPO_ART_PRODUCTO_DES
,SECCION_PRODUCTO_CVE
,SECCION_PRODUCTO_DES
,DIRECCION_PRODUCTO_CVE
,DIRECCION_PRODUCTO_DES
,COLOR_PRODUCTO_CVE
,COLOR_PRODUCTO_DES
,TALLA
,CLASE_PRODUCTO_CVE
,CLASE_PRODUCTO_DES
,FCH_CARGA
)
AS
SELECT
       VTA.TICKET
      ,VTA.MONTO_TICKET
      ,VTA.TIPO_PAGO_TICKET_CVE
      ,PAGO.TIPO_PAGO_TICKET_DES
      ,VTA.FCH_TICKET
      ,VTA.TIENDA_CVE
      ,TDA.TIENDA_DES
      ,VTA.PRODUCTO_CVE
      ,PROD.PRODUCTO_DES
      ,VTA.CANTIDAD_PRODUCTO
      ,VTA.MONTO_PRODUCTO
      ,PROD.FCH_PRODUCTO_ALTA
      ,PROD.PROVEEDOR_CVE
      ,PROD.PROVEEDOR_NOMBRE
      ,PROD.PRECIO_COSTO
      ,PROD.PRECIO_VENTA
      ,PROD.GPO_ART_PRODUCTO_CVE
      ,PROD.GPO_ART_PRODUCTO_DES
      ,PROD.SECCION_PRODUCTO_CVE
      ,PROD.SECCION_PRODUCTO_DES
      ,PROD.DIRECCION_PRODUCTO_CVE
      ,PROD.DIRECCION_PRODUCTO_DES
```

```
      ,PROD.COLOR_PRODUCTO_CVE
      ,PROD.COLOR_PRODUCTO_DES
      ,PROD.TALLA
      ,PROD.CLASE_PRODUCTO_CVE
      ,PROD.CLASE_PRODUCTO_DES
      ,CURRENT_DATE AS FCH_CARGA
FROM  DWH.FAC_VENTAS VTA
LEFT JOIN   SEM.FAC_PRODUCTO_TMP PROD
ON    VTA.PRODUCTO_CVE = PROD.PRODUCTO_CVE
LEFT JOIN   DWH.CATA_TIENDA TDA
ON    VTA.TIENDA_CVE = TDA.TIENDA_CVE
LEFT JOIN   DWH.CATA_TIPO_PAGO_TICKET PAGO
ON    VTA.TIPO_PAGO_TICKET_CVE = PAGO.TIPO_PAGO_TICKET_CVE
WHERE VTA.FCH_TICKET = CURRENT_DATE - 1
;
```

En el código de la vista se observa que se hacen cruces entre la tabla de ventas, con la tabla temporal creada en el bteq analizado anteriormente, además de los catálogos de tiendas y del tipo de pago. Además, sólo se hace por las ventas del día anterior de la carga, esto es ya que el DWH siempre carga la información del día anterior, este filtro de fecha se puso en el script de la vista, ya que si se desea hacer un reproceso de otra fecha es muy fácil el modificarlo y no es necesario hacer ningún cambio extra en el proceso.

Estos scripts se ejecutan por medio de shells job0600x.sh, al igual que los shells anteriores, se mandan los parámetros necesarios para la ejecución del script, nuevamente el nombre de este script es la unión del número del proceso y la tabla destino, en este caso sería *job06107_FAC_VENTAS.btq*. A continuación, se pone el registro del archivo de configuración con el cual se hace la carga del proceso de productos

```
job06107,1,sem,,area_sujeto,,,,SEM,FAC_VENTAS,VISTAS_SEM,VFAC_VENTAS
```

En el registro los parámetros que toma el proceso son el número de proceso, la etapa del proceso 1 (siempre va este valor), la fase del proceso de carga, el área sujeto a la cual vamos a cargar la información, el nombre de la base de datos y la tabla donde se va a cargar la información y por último el nombre de la base de datos y la tabla origen de la información. Analizando el log, que ejecuta este proceso se puede ver como toma los parámetros del registro anterior.

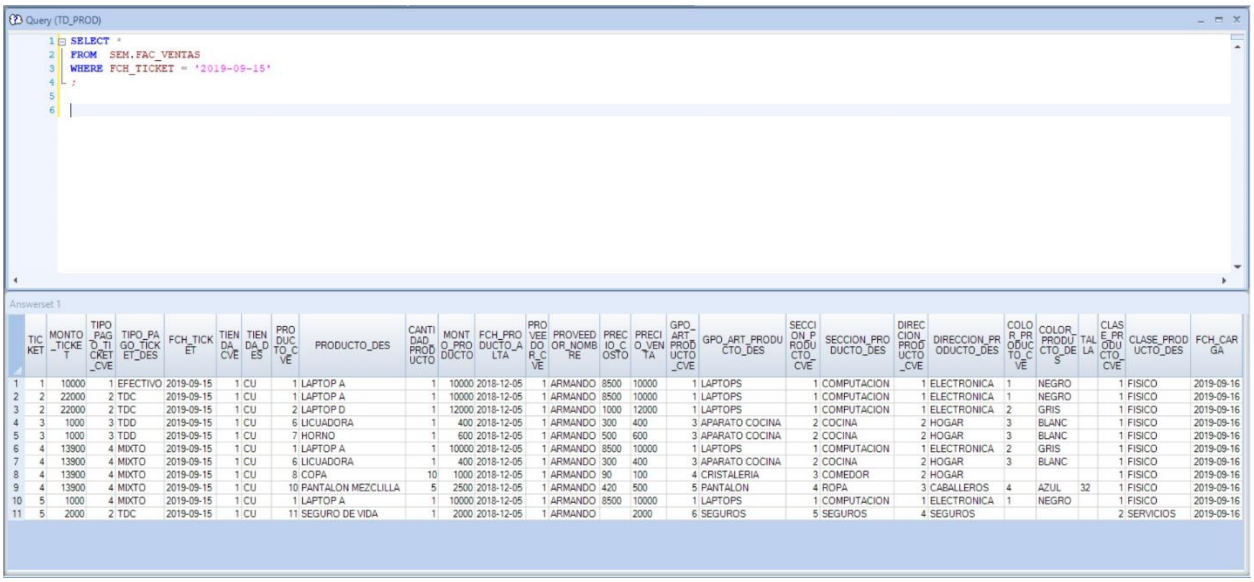
```
 _____________________________________________________________________________
|                                                                             |
|                                                                             |
|                                                                             |
|  ========== ESTE JOB ES EJECUTADO CON     D A T A  S T A G E ==========    |
|                                                                             |
|                                                                             |
|_____________________________________________________________________________|
El ambiente es:  Produccion
Comienza ejecución Mon Sep 16 09:43:41 CDT 2019
_____________________________________________________________________________

     Proceso de Ejecucion para JOB: 06107
_____________________________________________________________________________


_____________________________________________________________________________

    Registrando ejecucion en bitacora ini ...
_____________________________________________________________________________

*--- Carga de ejecucion ini exitosa  ---*
_____________________________________________________________________________

   Realizando ejecucion de carga a SEM ...
_____________________________________________________________________________

      INSERT INTO SEM.FAC_VENTAS
            SELECT * FROM VISTAS_SEM.VFAC_VENTAS;

 *** Insert completed. 659308 rows added.
 *** Total elapsed time was 1 minute and 23 seconds.


*--- Carga de ejecucion exitosa  ---*
Registros Insertados: 659308
Registros Actualizados: 0
Registros Borrados: 0
_____________________________________________________________________________

    Registrando ejecucion en bitacora fin ...
_____________________________________________________________________________

*--- Carga de ejecucion fin exitosa  ---*
Finaliza ejecución Mon Sep 16 09:48:10 CDT 2019
```

En esta etapa de carga, el posible mensaje de error que pueden ser cargado en la tabla de bitácora una vez que el proceso ha terminado, es el siguiente:

- Error en la Carga. Este error es cuando hubo algún problema en la carga, como puede ser un error de formato en una columna, que lleguen columnas diferentes a las que se espera en la tabla, por overflow o por llave primaria duplicada.

Los shells de carga a Semántica se ejecutan por medio de un job de DataStage de tipo Sequence, el cual realiza una malla interna de estos procesos, este job ya se ha explicado anteriormente. Este job de DataStage es ejecutado por medio de un shell job1000x.sh.

Para este proceso en particular, se puede observar que se hizo una integración de los datos, que ayuda a varias áreas usuarias sobre el comportamiento de las ventas de la empresa de Retail, sin necesidad de hacer un procesamiento en SAS o MicroStrategy muy grande, ya que la tabla que se procesó en la Semántica contiene la información necesaria. En la figura 23 se observa como los datos de esta tabla se despliegan.



TICKET	MONTO_TICKET	TIPO_PAGO_TICKET_CVE	TIPO_PAGO_TICKET_DES	FCH_TICKET	TIENDA_CVE	TIENDA_DES	PRODUCTO_CVE	PRODUCTO_DES	CANTIDAD_PRODUCTO	MONTO_PRODUCTO	FCH_PRODUCTO_ALTA	PROVEEDOR_CVE	PROVEEDOR_NOMBRE	PRECIO_COSTO	PRECIO_VENTA	GPO_ART_PRODUCTO_CVE	GPO_ART_PRODUCTO_DES	SECCION_PRODUCTO_CVE	SECCION_PRODUCTO_DES	DIRECCION_PRODUCTO_CVE	DIRECCION_PRODUCTO_DES	COLOR_PRODUCTO_CVE	COLOR_PRODUCTO_DES	TALLA	CLASE_PRODUCTO_CVE	CLASE_PRODUCTO_DES	FCH_CARGA
1	1	10000	1 EFECTIVO	2019-09-15	1	CU	1	LAPTOP A	1	10000	2018-12-05	1	ARMANDO	8500	10000	1	LAPTOPS	1	COMPUTACION	1	ELECTRONICA	1	NEGRO		1	FISICO	2019-09-16
2	2	22000	2 TDC	2019-09-15	1	CU	1	LAPTOP A	1	10000	2018-12-05	1	ARMANDO	8500	10000	1	LAPTOPS	1	COMPUTACION	1	ELECTRONICA	1	NEGRO		1	FISICO	2019-09-16
3	2	22000	2 TDC	2019-09-15	1	CU	2	LAPTOP D	1	12000	2018-12-05	1	ARMANDO	1000	12000	1	LAPTOPS	1	COMPUTACION	1	ELECTRONICA	2	GRIS		1	FISICO	2019-09-16
4	3	1000	3 TDD	2019-09-15	1	CU	6	LICUADORA	1	400	2018-12-05	1	ARMANDO	300	400	3	APARATO COCINA	2	COCINA	2	HOGAR	3	BLANC		1	FISICO	2019-09-16
5	3	1000	3 TDD	2019-09-15	1	CU	7	HORNO	1	600	2018-12-05	1	ARMANDO	500	600	3	APARATO COCINA	2	COCINA	2	HOGAR	3	BLANC		1	FISICO	2019-09-16
6	4	13900	4 MIXTO	2019-09-15	1	CU	1	LAPTOP A	1	10000	2018-12-05	1	ARMANDO	8500	10000	1	LAPTOPS	1	COMPUTACION	1	ELECTRONICA	2	GRIS		1	FISICO	2019-09-16
7	4	13900	4 MIXTO	2019-09-15	1	CU	6	LICUADORA	1	400	2018-12-05	1	ARMANDO	300	400	3	APARATO COCINA	2	COCINA	2	HOGAR	3	BLANC		1	FISICO	2019-09-16
8	4	13900	4 MIXTO	2019-09-15	1	CU	8	COPA	10	1000	2018-12-05	1	ARMANDO	90	100	4	CRISTALERIA	3	COMEDOR	2	HOGAR				1	FISICO	2019-09-16
9	4	13900	4 MIXTO	2019-09-15	1	CU	10	PANTALON MEZCLILLA	5	2500	2018-12-05	1	ARMANDO	420	500	5	PANTALON	4	ROPA	3	CABALLEROS	4	AZUL	32	1	FISICO	2019-09-16
10	5	1000	4 MIXTO	2019-09-15	1	CU	1	LAPTOP A	1	10000	2018-12-05	1	ARMANDO	8500	10000	1	LAPTOPS	1	COMPUTACION	1	ELECTRONICA	1	NEGRO		1	FISICO	2019-09-16
11	5	2000	2 TDC	2019-09-15	1	CU	11	SEGURO DE VIDA	1	2000	2018-12-05	1	ARMANDO		2000	6	SEGUROS	5	SEGUROS	4	SEGUROS				2	SERVICIOS	2019-09-16

Figura 23. Tabla de la Semántica de las ventas (Asistente de Teradata)

Como se puede apreciar, en cada una de las etapas de carga se hicieron pruebas unitarias, dividiéndose en varios pasos, ya que en el primer paso se hicieron con datos inventados, para probar el funcionamiento de los procesos, este paso no aplicó con los procesos de extracción de datos. La segunda etapa se realizó con datos proporcionados por los sistemas fuentes de sus ambientes de prueba, para las procesos de extracción aquí se comenzaron con las pruebas unitarias.

Pruebas integrales

Las pruebas integrales se hicieron atendiendo dos rubros distintos, la primera es atendiendo pruebas técnicas y la segunda es atendiendo pruebas de negocio. Las pruebas técnicas se hicieron con base en la matriz de pruebas de la tabla 7.

PRUEBA	ENTRADAS	OBSERVACIONES	RESULTADO
Prueba de volumetría	Se hace con datos productivos	Se validan los datos de entrada vs los de carga en las tablas en el DWH	Los datos son los esperados
Prueba de tiempo de procesamiento	Se ejecutan los procesos del DWH	Se valida el tiempo que tardan los procesos en ejecutarse	El tiempo de procesamiento es muy bueno en comparación con el DWH anterior
Prueba de estrés	Se ejecutan todos los procesos al mismo momento y se hacen consultas a algunas tablas creadas	Se valida el tiempo de respuesta de las tablas consultadas, así como el tiempo que tardan los procesos en relación con la prueba anterior	El tiempo de procesamiento no aumenta mucho comparando los tiempos de la prueba anterior
Prueba de errores	Se ejecutan los procesos con errores simulados	Se hace fallar a los procesos con distintos errores que puedan suceder en los procesos de carga	Se valida que cada error el proceso de carga lo pueda identificar y terminar el proceso de forma errónea y que el log de carga identifique el error correcto
Prueba de re-ejecución de proceso	Se re-ejecutan los procesos después de un error	Se hace fallar a los procesos y se ejecuta de nuevo para ver su comportamiento	Se valida que los procesos se reinicien a partir del punto en donde se detectó el error
Prueba de reinicio de los procesos del DWH	Se ejecuta el proceso de reinicio de procesos	Se ejecuta el proceso con el que se reinician los jobs de DataStage	Se valida que al ejecutar el proceso de reincio de los jobs de DataStage, los jobs estén en un estado compilado, sin importar si terminaron bien o no

Tabla 7. Matriz de pruebas técnicas (Diseño propio)

Después de obtener el visto bueno de la matriz de pruebas técnicas por parte del área encargada del DWH en la empresa de Retail, se hizo una segunda matriz de pruebas, pero a nivel de negocio. Estas pruebas se hicieron con base en la matriz de pruebas de la tabla 8.

PRUEBA	ENTRADAS	OBSERVACIONES	RESULTADO
Prueba de carga de un día de batch	Se hace con datos productivos	Se validan los datos de entrada vs los de carga en las tablas en el DWH	Se valida que los datos cumplan con todas las reglas de negocio
Prueba de carga de un segundo día de batch	Se hace con datos productivos	Se validan los datos de entrada vs los de carga en las tablas en el DWH	Se valida que los datos sigan cumpliendo con las reglas de negocio, después de que las tablas tengan una nueva carga de información

Tabla 8. Matriz de pruebas de negocio (Diseño propio)

Por último, al obtener el visto bueno de la matriz de pruebas de negocio por parte de las área usuarias, se comenzó con la preparación del pase a producción del DWH. Cómo se observa en las matrices de prueba, las validaciones se hicieron pensando en todos los aspectos en la carga diaria del DWH.

Puesta en producción y soporte

Una vez que se terminaron las pruebas integrales y se tuvo el visto bueno de la empresa de Retail, se programó el pase a producción del DWH en varias fases, esto debido a la cantidad de procesos que se debían de pasar, las fases se dividieron de la siguiente manera

- Pase a producción de los componentes. En un día se pasaron todos los componentes creados del DWH, a nivel de base de datos y del sistema operativo.
- Cargas históricas. Estas cargas se hicieron en dos partes, la primera fue la carga de toda la información en una semana, la segunda parte se hizo a la

siguiente semana con la información de la semana faltante. Esta última parte se hizo el mismo día en que se activo el DWH en Control – M, para que comenzara su ejecución de forma diaria.

- Carga diaria. Se comenzó con esta carga al segundo día de ejecución de los procesos del DWH en Control – M, validando que ya sólo enviaran los sistemas fuente la información sin cargas históricas.

Las siguientes dos semanas se tuvieron de soporte para resolver posibles problemas que los procesos tuvieran en su carga diaria, así como posibles errores de carga por escenarios que no se tenían contemplados en la construcción, en la definición de los procesos y que no surgieron en las pruebas integrales.

Finalmente, el arquitecto de la base de datos y yo, tuvimos sesiones con la gente encarga del DWH de la empresa de Retail, para darles una capacitación de los procesos de carga y para el mantenimiento a los mismos.

Resultados

El nuevo DWH quedó con mejoras en los procesos de carga, las cuales fueron integradas en cada etapa del proceso, aún en las tablas que se nos pidió que debían de quedar con la misma estructura, se pudieron hacer algunos cambios con los que se mejoró el rendimiento de las tablas y las cargas de las mismas.

Además, la forma en que se diseño tiene una mejor integración en todo el DWH que el anterior, esto debido a que las tablas fueron creadas de forma que, si se tuvieran nuevos datos en un futuro, las tablas fueran capaces de no tener que modificar mucho su estructura, lo cual en un DWH es algo que siempre debe de considerarse.

Esto se tomó en cuenta, debido a que el comportamiento de compra de los clientes va cambiando conforme pasa el tiempo, ya que hace unos años las compras en línea con respecto a las compras físicas eran una cantidad muy pequeña, pero en estos tiempos esas compras ya tienen un rol muy importante dentro de las empresas de Retail, ya que hoy en día hay empresas que sólo venden en línea.

Por lo que se tuvo mucho cuidado en el diseño de las tablas del módulo de las ventas, para que pudieran integrar en un futuro las compras en línea, dichas compras después de algunos años fueron integradas a este módulo, con pequeñas modificaciones a las tablas y a sus respectivos procesos de carga.

El producto que entregué con el equipo, cumplió con todos los requerimientos que la empresa de Retail nos solicitó, aún cuando en un principio se tuvieron algunas complicaciones con el inicio del mismo. Sin embargo, al entregar el DWH el cliente quedó muy satisfecho con todo el proceso, así como la forma en que quedaron estructurados los datos.

Debido a la forma en que diseñé el proceso de carga del DWH, cuando se deba de integrar un nuevo módulo, sólo es necesario preocuparse por las nuevas reglas de negocio, ya que los procesos están construidos con base en parámetros, para que la modificación a los shells sea mínima, así como a los scripts de tipo btq y tpt vistos anteriormente. Solamente debe de insertarse en la tabla de usuarios y/o en la tabla de procesos, los nuevos usuarios y procesos para que funcionen en las nuevas ejecuciones del batch del DWH, cómo se muestra en la figura 24.

Figura 24. Insert de nuevos usuario y procesos del DWH (Asistente de Teradata)

Cuando hice la capacitación del proceso batch del DWH en conjunto con el arquitecto de BD, la empresa de Retail nos pidió una forma sencilla en que se pudiera dar el seguimiento a todo el proceso, ya que para todas las áreas involucradas era un proceso nuevo, por lo que al momento de comenzar con su monitoreo se les hacía difícil intentar corregir el problema.

Por lo que se me ocurrió realizar un archivo de monitoreo en Excel, con todos los procesos de carga del DWH, conectándose a la tabla de bitácora del DWH (explicada anteriormente), por lo que de forma gráfica se pudiera hacer un seguimiento de la carga diaria. Esta propuesta fue muy bien recibida por parte de la empresa de Retail, ya que era una forma muy sencilla de enseñar el proceso en su ejecución diaria. En la tabla 9 se puede observar parte del archivo de monitoreo (nuevamente los nombres y procesos mostrados son de ejemplo).

EXTRACCIÓN				STG				DWI			DWH	
JOB	FUENTE	JOB EXT	TABLA	JOB	FUENTE	JOB STG	TABLA	JOB	JOB DWI	TABLA	JOB DWH	TABLA
				job10004	VENTAS	job02006	PAGO	job10007	job04002	PAGO	job05002	PAGO
						job02013	TIPO_TRAN		job04003	TRAN	job05003	TRAN
						job02014	TRAN		job04004	CUENTA	job05004	CUENTA
						job02015	FAC_PRMCRM					
						job02016	TIPP_PAGO					
						job02017	CUENTA					
						job02018	TIPO_CUENTA					
job10005	INVENTARIO	job01015	INVENTARIO_01	job10006	INV_STG	job02005	INVENTARIO	job10035	job04001	INVENTARIO	job05001	INVENTARIO
		job01019	INVENTARIO_02									
		job01064	INVENTARIO_03									
		job01065	INVENTARIO_04									
		job01066	INVENTARIO_05									

Tabla 9. Archivo de monitoreo de los procesos del DWH (Diseño propio)

La tabla 9 se construyó con dos procesos, el primero es el de las ventas y el segundo es el del inventario. El archivo se hizo de forma en que se pueda ver la dependencia y el proceso sucesor, en el primer caso el job10004 es el proceso de carga a Staging de las ventas y le da dependencia al proceso job10007 que es la carga al DWH de las ventas, en el segundo caso el job10005 es el proceso de extracción del inventario, le da condición al proceso job10006 que es la carga a Staging y este proceso le da condición al proceso job10035 que es la carga al DWH.

Este archivo también se actualiza cada vez que un nuevo proceso en el DWH es integrado, esta actividad la hace el área encargada del DWH de la empresa de Retail, ya que el monitoreo de los procesos batch es su responsabilidad. Cuando esto sucede

entrego una relación de los nuevos procesos creados, tomando en cuenta la tabla 9 y el proceso de inventario, la relación entregada sería la siguiente

- Para el proceso de Extracción. El proceso general es el *job10005*, su fuente es *INVENTARIO*, los procesos internos que ejecuta son el *job01015* que ejecuta la extracción de la tabla de *INVENTARIO_01*, el *job01019* que ejecuta la extracción de la tabla de *INVENTARIO_02*, el *job01064* que ejecuta la extracción de la tabla de *INVENTARIO_03*, el *job01065* que ejecuta la extracción de la tabla de *INVENTARIO_04* y el *job01066* que ejecuta la extracción de la tabla de *INVENTARIO_05*.
- Para el proceso de Staging. El proceso general es el *job10006*, su fuente es *INV_STG*, el proceso interno es el *job02005* que carga a la tabla de Staging de *INVENTARIO*.
- Para el proceso de DWH. El proceso general es el *job10035*, el proceso interno de DWI es el *job04001* que carga a la tabla de *INVENTARIO* y el proceso interno de DWH es el *job05001* que carga a la tabla de *INVENTARIO*.

Por último, gracias a la facilidad del monitoreo del batch y al proceso de carga del DWH, la integración de nuevos miembros en el equipo de desarrollo ó en el equipo de la empresa de Retail no es complicada, aún cuando no tengan conocimientos en todas las herramientas que se usan en el DWH.

Conclusiones

El nuevo DWH quedó con las mejoras propuestas en los procesos de carga, las cuales fueron integradas en cada etapa del proceso, mejorando el rendimiento de las tablas, las cargas y el mantenimiento de las mismas, ya que se tuvo que realizar una actividad previa a la construcción del DWH, por lo que en lugar de comenzar con el análisis y el desarrollo del DWH como se tenía contemplado, primero se ayudó con la obtención de los datos que lo alimentarían. Esta actividad fue muy importante porque gracias a esto, se pudo tener un conocimiento a nivel negocio de cómo funciona una empresa de Retail, pudiendo hacer mejoras en el diseño del DWH. A continuación, en la tabla 10 se pone las actividades y los tiempos finales que se hicieron en este proyecto.

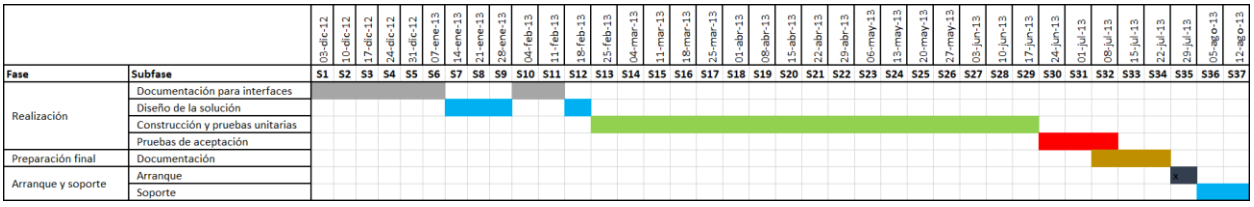


Tabla 10. Cronograma final de actividades (Diseño propio)

El proceso que se tomó de ejemplo en las distintas etapas de carga en el DWH está representado en la figura 25, donde a partir de la información que se tiene en distintos sistemas fuentes, gracias al diseño del DWH se integró en una sola tabla en la Semántica facilitando el análisis de esa información.

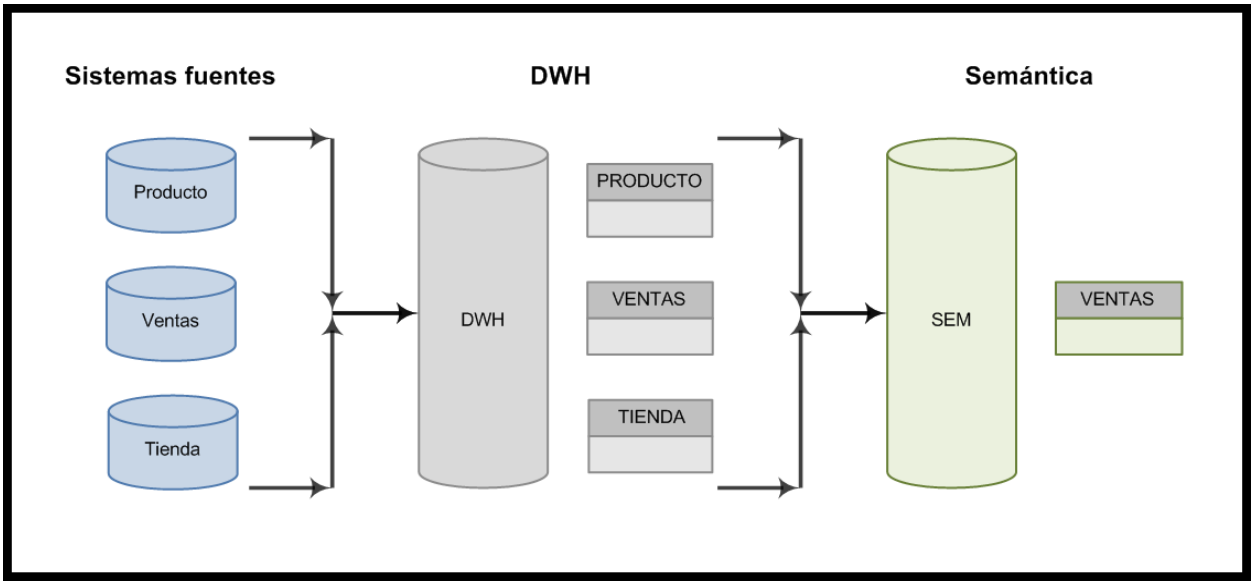


Figura 25. Integración de los datos en el DWH (Diseño propio)

Gracias a estas mejoras y los resultados obtenidos, se logró hacer una muy buena relación entre mi consultora y la empresa de Retail obteniendo su confianza, ya que obtuvimos muy buenas respuestas con las distintas áreas usuarias que empezaban a consultar el DWH y con el área técnica encargada del DWH.

Para concluir este reporte, la confianza obtenida desde ese momento se ha mantenido vigente después de 6 años, ya que se sigue desarrollando, modificando y dando soporte al DWH. Siendo una parte medular dentro de la empresa de Retail, debido a toda la información que actualmente se procesa, ya que más áreas dentro de la empresa han empezado a usar el DWH desde que se hizo esta primera fase, haciendo que el Data Warehouse siga creciendo gracias a los nuevos requerimientos que todas las áreas continúan pidiendo.

Anexo

1. Retail. Es la industria de venta minorista o venta al detalle para cubrir las necesidades de los consumidores finales, como por ejemplo los supermercados. Teniendo tres tipos de empresa de este tipo.

   Offline. Realizan su actividad comercial de ventas de manera física o presencial.
   E-Retail. Realizan su actividad comercial en Internet, también llamado como Ecommerce.
   Híbrido. Realizan su actividad comercial de manera física y en Internet

2. DataStage. Es una herramienta ETL que utiliza una notación gráfica para construir soluciones de integración de datos y está disponible en varias versiones, como Server Edition y Enterprise Edition. Es usado en una arquitectura cliente–servidor, además el servidor puede implementarse en Windows como en Unix, siendo una herramienta muy poderosa de integración de datos muy usada en proyectos de DWH.

3. Proceso batch. Se conoce como proceso batch, a la ejecución de un programa sin el control o supervisión directa del usuario. Este tipo de programas se caracterizan porque su ejecución no precisa ningún tipo de interacción con el usuario. Generalmente, este tipo de ejecución se utiliza en tareas repetitivas sobre grandes

conjuntos de información, ya que sería tedioso y propenso a errores realizarlo manualmente.

4. File system. Es el componente del sistema operativo encargado de estructurar la información guardada en una unidad de almacenamiento, así como la administración del espacio libre y del acceso a los datos resguardados.

5. DB Link. Objeto creado en una base de datos que sirve para conectarse a otra base de datos remota, generalmente para hacer consultas a las tablas de esa base de datos remota, aunque también puede ser para realizar actualizaciones o borrado de datos en incluso modificaciones a las estructuras de las tablas.

6. Stage. Es el nombre denominado a cada componente dentro de un Job de DataStage, sin importar el tipo de Job que se esté usando.

7. Llave subrogada. Una llave subrogada es un identificador único que se asigna a cada registro de una tabla. Esta clave generalmente no tiene ningún sentido específico de negocio, siendo de tipo numéricas y auto incrementales.

   Estas llaves sirven para hacer cruces entre tablas, ya que es más rápido el cruzar por campos de tipo numérico que de tipo alfanumérico, son usadas comúnmente en los DWH.

Bibliografía

Data Waterhouse conceptos, recuperado de

https://www.powerdata.es/data-warehouse

Teradata Parallel Transporter User Guide, recuperado de

https://developer.teradata.com/sites/all/files/documentation/linked_docs/2445020A_TPT-User-Guide-13.10.pdf

Basic Teradata Query Reference, recuperado de

https://developer.teradata.com/sites/all/files/documentation/linked_docs/2414020A_BTEQ-Reference-13.10.pdf

IBM InfoSphere DataStage Server Job Developer's Guide, recuperado de

https://www-05.ibm.com/e-business/linkweb/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=SC18-9898-03

Conceptos de Retail, recuperado de

https://www.definicionabc.com/economia/retail.php

Conceptos de la metodología Agile, recuperado de

https://openwebinars.net/blog/que-es-la-metodologia-agile/

Base de Datos Teradata: Introducción Técnica.pdf